Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Management Science and Information Systems Studies

Final Year Project Report

Wildlife Rescue Management System: Developing Mobile and Web-based Applications for a Charitable Organization

Jack Cleary

April 2023

TRINITY COLLEGE DUBLIN

Management Science and Information Systems Studies Project Report

PEARSE STOKES

Wildlife Rescue Management System: Developing Mobile and Web-based Applications for a Charitable Organization

April 2023

Prepared by: Jack Cleary   Supervisor: Alessio Benavoli

Declaration

I declare that the work described in this dissertation has been carried out in full compliancewith the ethical research requirements of the School of Computer Science and Statistics.

I have read and I understand the plagiarism provisions in the General Regulations of theUniversity Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write',located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

I declare that the report being submitted represents my own work and has not been taken fromthe work of others save where appropriately referenced in the body of the assignment.

Signed: Jack Cleary .

Jack Cleary 09/04/2023

ABSTRACT

The objective of this project was to research, design and develop a mobile application and a web-based application for wildlife rescue charities, in Ireland and internationally, which would allow users to communicate effectively with each other in an internal messaging system, as well as log and respond to cases of animals requiring attention. The project involved identifying the system requirements and developing an intuitive solution which promoted usability to meet the requirements. The system was developed using JavaScript, JSX, React, React Native, Expo and Firebase.

PREFACE

This project was undertaken on behalf of Pearse Stokes (the Client) and the many wildlife charities of which he is a generous supporter, with the aim of providing an application for volunteers and administrators alike. The project involved the design and development of an intuitive, user-friendly platform which allows users to view and respond to wildlife rescue cases, as well as to communicate with the other volunteers and administrators.

The system developed met all the requirements agreed with the client, and also included features and functionalities which have expanded on the original terms of reference. The system has been designed to provide an intuitive solution to the user, which takes a minimalistic approach and promotes the system’s accessibility and usability. A consistent layout and simple components were used throughout the design to promote user recall and provide an easily traversable system.

The project has successfully met the terms of reference as agreed with the client, albeit with some difficulty. The most notable challenge encountered was that the developer had no prior experience with the technologies and frameworks used to develop the system, and hence a steep learning curve was faced from the outset. Despite this, a fully functioning system has been sent to the client, ready for deployment.

I would like to thank the client, Pearse Stokes, for his input and enthusiasm throughout the timeframe of this project, and for his assistance in the creation of the system. Without his generous support, a system such as this may never have been developed.

I would also like to thank my supervisor, Alessio Benavoli for his support and guidance throughout the project, and all his faculty colleagues in Trinity College Dublin who have passed on their knowledge and expertise to myself and countless others.

Finally, I would like to thank the residents of 60 Saint Columbanus’ Road, past and present, for creating a living environment focused on motivation and encouragement, which was of great benefit to me over the course of this project.

Wildlife Rescue Management System: Developing Mobile and Web-based Applications for a Charitable Organization
April 2023

TABLE OF CONTENTS

APPENDICES

1 INTRODUCTION & SUMMARY

This section of the report introduces the client, the background information regarding this project and details the agreed upon terms of reference. A summary of remaining chapters of this report is also provided in this section.

1.1 The Client

The client is Pearse Stokes, who has been involved in Wildlife Rescue Organisations for 7 years, in recent times primarily with Kildare Wildlife Rescue. Over that time he has transformed many of the processes and protocols used in Wildlife Rescue with a view to professionalising the sector. The client's mission is to aid these organisations in protecting and rescuing vulnerable wildlife in Ireland. This project aligns with their core mission as it pertains to improving the efficiency with which volunteers and administrators can fulfil their roles.

It should be noted that the client also runs a graphic design firm in Dublin, and as a result he instructed that focus should be on the functionality of the application, rather than on the aesthetics as these could be improved upon at a later date.

1.2 Project Background

Ireland is home to an wide variety of different creatures both land and sea, with over 50 mammals and 400 species of bird. From herds of red deer in Killarney National Park to wild ponies in Connemara – Ireland's national parks and nature preserves offer a feast for the eyes of every animal and nature lover.

Wildlife rescue charities play a critical role in Ireland's biodiversity conservation efforts. With such a diverse range of wildlife, including many endangered species, Irish wildlife rescue charities help to protect and conserve these species by providing rehabilitation and care for injured, sick or orphaned animals. The work of these charities is vital in ensuring that these animals have the best possible chance of survival and the opportunity to thrive.

In addition to their conservation efforts, wildlife rescue charities also play an important role in educating the public about the importance of wildlife preservation and the need to protect and conserve our natural environment. They help to raise awareness of the challenges facing Ireland's wildlife and the steps that can be taken to address these challenges, such as reducing plastic waste and protecting natural habitats.

Currently, these charities are run and organised, albeit by extremely hard-working and selfless people, using the very inefficient system that is WhatsApp. An administrator in the Wildlife Rescue Centre would receive a phone call from a member of the public, providing details of an animal in need of rescue. These details are then sent into (often more than one) WhatsApp group, where volunteers can respond, or not respond, to the case. This project aims to bring a fresh approach to these organisations by way of a centralised and internalised application, through which volunteers can view details of, and respond to, cases of wildlife requiring rescue. It will also enable them to communicate effectively with other members of the organisation through an internal chat messaging system.

At a high level, the project involves the following:

- Development of a web-based dashboard, to be used by administrators, where they can add cases to a database.

- Development of a mobile application for volunteers to use, through which they can view open cases and choose to respond or not. This application should be operable on both iOS and Android devices.
- Development of a chat messaging system, accessible from both the mobile and web applications, through which administrators can release urgent messages and volunteers can effectively communicate with each other.
- Ensuring that any systems developed are free of charge, or at least low-costing, as this charity will be unable to afford expensive fees.

1.3 Terms of Reference

The agreed upon terms of reference for the project are detailed below:

- Using free and open-sourced technologies, design, develop and implement a web-based solution which allows administrators to effectively log cases of wildlife requiring assistance.
- Using free and open-sourced technologies, design, develop and implement a mobile application operable on both iOS and Android platforms, which allows volunteers to view and respond to cases of wildlife requiring assistance.
- Develop an internal chat messaging application, built into both the web-based and mobile applications, so that administrators and volunteers alike can communicate more efficiently.
- Store all cases and all chat messages in a database for review and statistical analysis.
- Ensure that the system is developed free of charge, or at least at low-cost, in order to avoid the accrual of expensive fees for the charity organisation

It should be noted that the client does not yet require, or want, this system to be deployed. This project serves as a conceptualisation of how this system could be designed and developed, but is not intended for use until a later date.

1.4 Summary of Remaining Chapters

- Chapter 2 provides a system overview. It includes an overview of the system's goals, a synopsis of the technical environment, a description of how the system's primary features work, and a sitemap.
- Chapter 3 provides a summary of the work produced throughout the project. It provides information on the design and development phases that went into creating the finished system. Determining the system requirements, examining and selecting appropriate technologies, designing the system, developing the system, and testing the system are just some of these processes. This chapter also includes a list of the major obstacles encountered.
- Chapter 4 includes a variety of proposals for future advancements and a summary of the conclusions that may have been taken from the project.

2 DEMONSTRATION

This section of the report provides video demonstrations of the system in use. It is advisable that before continuing with this report, the reader should watch the attached videos, which illustrate the live use of the developed applications. These videos provide a comprehensive understanding of how the application operates in real-time and offers a valuable visual aid to complement the forthcoming system overview. The link below will navigate to a Google Drive where the videos can be viewed. Please begin by reading the very brief notes provided, before watching the videos themselves.

Link: Wildlife Rescue App Demonstrations

3 SYSTEM OVERVIEW

An overview of the completed system is provided in this chapter. It includes a summary of the system's goals, a synopsis of the technical environment, a description of how the system's primary features work, and a sitemap.

3.1 System Objectives

The objective of this system, as detailed in the terms of reference in Section 1.3, is to provide the client with a proof of concept for a solution that will allow volunteers and administrators in a wildlife charity to work more efficiently. This system will provide a central database which will store details of animals that are in need of rescue, as well as a web-based dashboard for administrators to use which will allow them to add cases to the database in an intuitive and straightforward manner. Administrators must be able to input specific information relating to the cases, i.e. species of animal, location etc, as well as attach photos and or videos to each case, all of which must be stored in the database for future statistical analysis.

A mobile application must also be designed, intended for the use of volunteers (who will be carrying out the actual rescuing). This app must display the cases stored in the database, along with a status - either active, in progress or rescued. It is vital that this mobile application is operable on both iOS and Android.

Users should be able to sign up and create an account for either the web-based application or the mobile app. There must also be a chat messaging service built-in to the system, to allow for administrators to communicate alerts to all volunteers, and for volunteers and administrators alike to communicate via a central forum.

As briefly explained above, the system will have two types of user, administrators and volunteers. Volunteers will only have access to the mobile application, and only administrators will have access to the web-based dashboard. Administrators will require verification before being allowed to access the web-app. Any user should be able to download the mobile app and sign up as a volunteer. However, if they have not completed the required training they will not be permitted to rescue any animals until they do so.

One important objective of this project is that the solution provided should be either completely free to use, or as low cost as possible. This is a very important point to highlight, as the charity are unable to afford to pay expensive hosting and storage fees. Hence, more efficient, but more expensive, software options were ruled out. The system provided is completely free to use, and will be so unless a certain threshold is reached with regard to the size of the database.

3.2 Technical Environment

The system has been developed using JavaScript, JSX, NodeJS, React Native, Expo CLI and Firebase along with some other software in the underlying architecture. The system has both a back end and front end structure, which may also be denoted as server-side and client-side respectively.

Firebase is a mobile and web application development platform that provides a suite of backend services to help developers build and manage their applications. It is a cloud-based platform that lets developers build and deploy scalable apps quickly and efficiently without having to worry about infrastructure management (Omisola, 2021). A number of backend services are offered by Firebase, such as real-time databases, authentication, cloud storage, hosting, and others. These services are made to be highly scalable and secure, so that developers can concentrate on creating applications and providing value to their users. The authentication service offers a safe and user-friendly approach to authenticate users, and the cloud storage service offers scalable and dependable storage for application data (Google, 2023). The version of the Firebase JavaScript library used in this project is '*firebase@9.19.1'.*

The front-end was developed using React Native and the Expo CLI. React Native is used to develop applications for a wide range of platforms by enabling developers to use the React framework along with native platform capabilities (ReactNative, 2023). In effect, it takes the React code you have written and converts it into code native to the platform on which it is being run. The version of React Native being used is '*react-native@0.71.6'* in conjunction with React version '*react@18.2'.*

A Firebase back-end integrates seamlessly with a React Native Expo front-end with a host of language specific methods that allow you to store data, query the database, authenticate users and much more with ease (AtomLab, 2022). No querying language was used as it was not necessary, all relevant querying could be done using methods from various libraries.

Further details pertaining to the process by which the technical environment was determined can be found in *Section 4.2*.

3.3 System Overview

A sitemap of the client-side system is illustrated by the figures below. *Figure 3.3.1* exhibits the web-based, administrator application, while *Figure 3.3.2* illustrates the volunteer's mobile application. The intent was for both solutions to be very intuitive, with little margin for human error, and a simplistic flow in order to make it as straightforward as possible for the end-user, many of whom may not be overly proficient with technology.

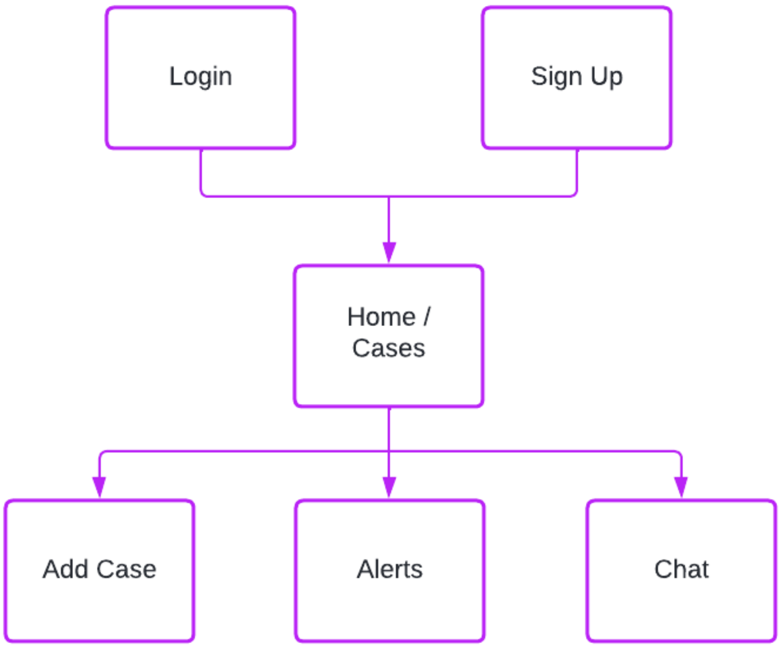


Figure 3.3.1 - WebApp Sitemap

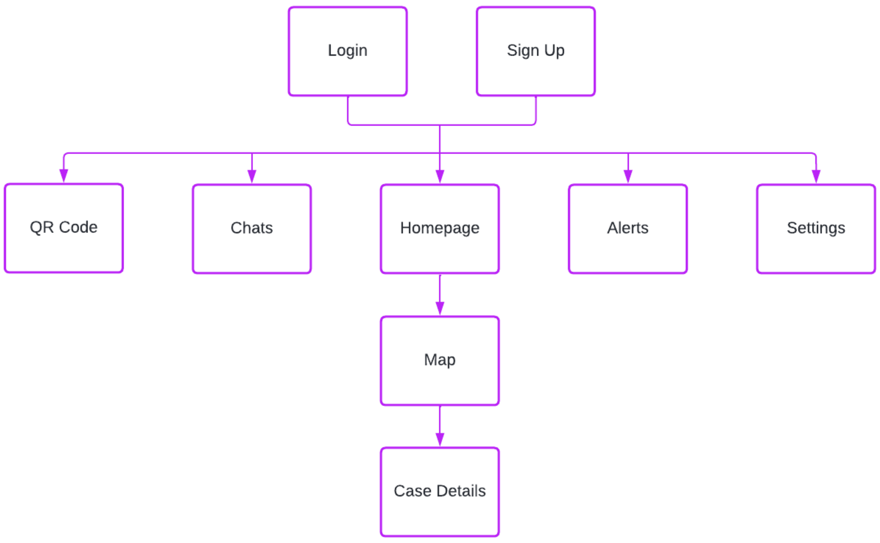
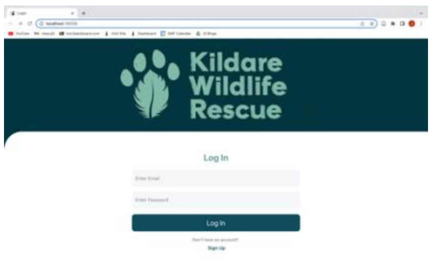
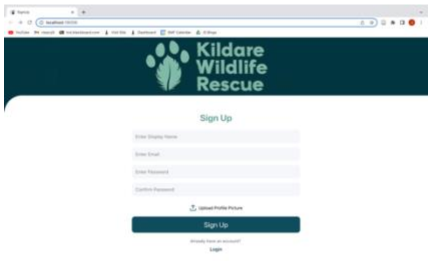


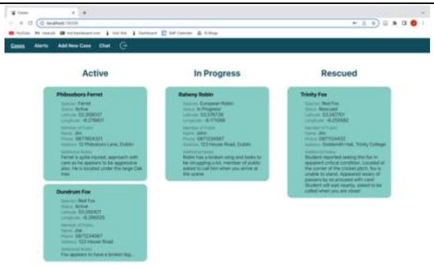
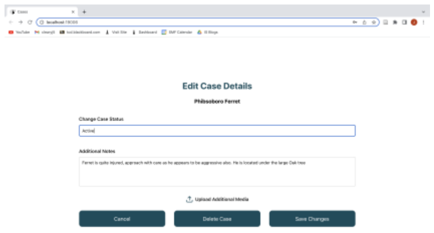
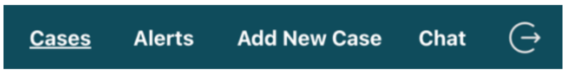
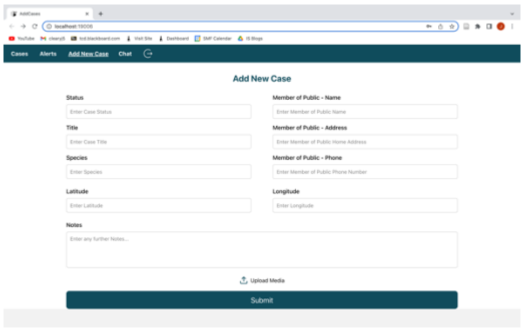
Figure 3.3.2 - Mobile App Sitemap

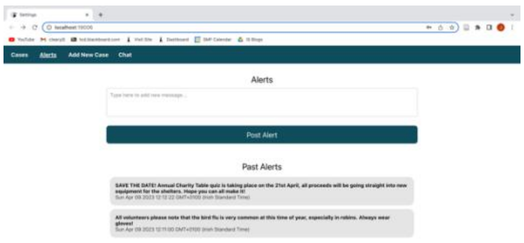
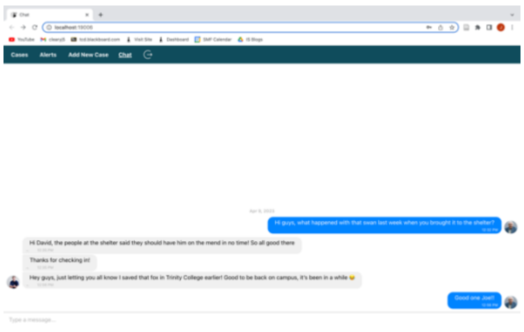
Below are descriptions of the pages that users will find in each application, first the web-based app and then the mobile app. These descriptions are accompanied by images of the relevant screens.

<u>Web-Based Dashboard</u>

Table 3.3.1 - Web App System Description & Screens

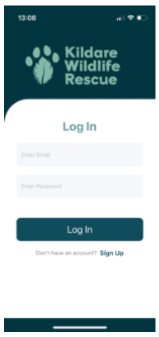
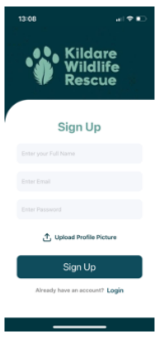
Description	***Image***
Login and Sign-Up Pages As stated in the System Objectives – Section 3.1 – users should be able to sign up and create accounts in order to use this application. The login and sign up pages are very similar screens with simplistic layouts. They contain a back Image with the charity logo, and a container overlaying this with a form inside. The form has fields which vary depending on which page you are viewing, for Login just email and password, whereas for Sign Up users are asked for a Display Name also, mainly for the messaging system, and to confirm their password. Below this form are buttons to upload a profile picture and to submit the details, either to log in or create the account. And lastly, there is also a piece of text with a button to change to the alternative page depending on whether the user already has an account created or not.	 *Figure 3.3.3 – Login Page*  *Figure 3.3.4 – Sign Up Page*

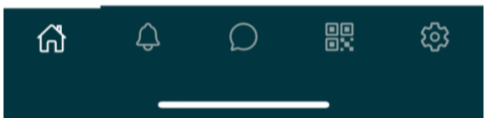
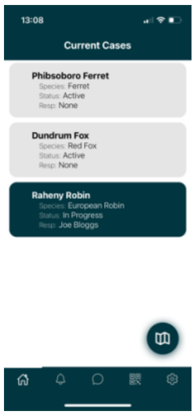
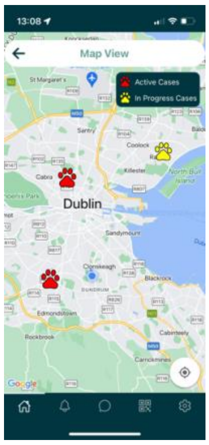
Home Screen The Home Screen, or landing screen, of the web-based application is where administrators can view and add cases of animals requiring rescue. The cases are divided into 3 columns (by status), either Active, In Progress or Rescued, as shown in *Figure 3.3.5*. These different statuses are detailed further in *Section 4.4*. In order to change or alter a casefile's details, the user can simply click on the case and change the text in the Text Input fields provided.	 Cases Alerts Add New Case Chat Active In Progress Rescued Phibsboro Ferret Raheny Robin Trinity Fox Dundrum Fox *Figure 3.3.5 – Home Page* Edit Case Details Phibsboro Ferret Change Case Status Active Additional Notes Ferret is quite injured, approach with care as he appears to be aggressive also. He is located under the large Oak tree Upload Additional Media Cancel Delete Case Save Changes *Figure 3.3.6 –Edit Details Pop*
Navigation Bar and Flow Each page of the web application contains the Navigation bar at the top, which allows the user to navigate fluidly between screens. The current screen is highlighted and underlined, as can be seen in the *Figure 3.3.7.*	 *Figure 3.3.7 – Web App Nav Bar*
Add Case The Add Case button in the Navigation bar will take the user to the screen shown in *Figure 3.3.8*. This is a simple form into which the administrator will enter the details of animals in need of rescue, as reported by members of the public via the phone/helpline. All relevant details can be entered, and photos/ videos attached if relevant, and then submitted to the database using the submit button at the bottom. The details will then appear on the Home Page in the relevant column.	Add New Case Status: Enter Case Status Member of Public - Name: Enter Member of Public Name Title: Enter Case Title Member of Public - Address: Enter Member of Public Home Address Species: Enter Species Member of Public - Phone: Enter Member of Public Phone Number Latitude: Enter Latitude Longitude: Enter Longitude Notes: Enter any further Notes... Cases Alerts Add New Case Chat Upload Media Submit *Figure 3.3.8 – Add Cases*

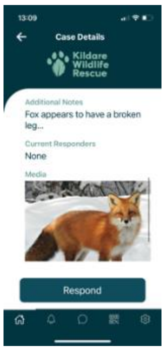
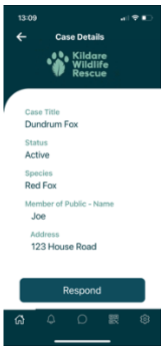
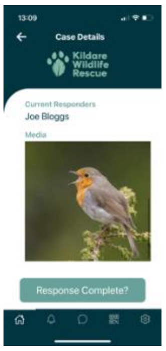
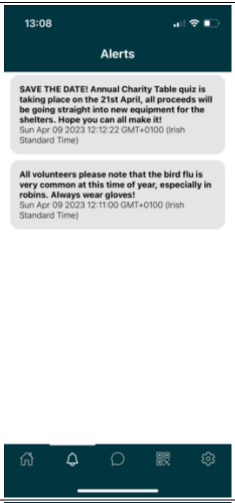
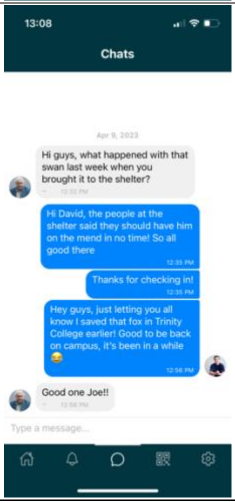
Alerts The function of this screen is for the administrators to communicate alerts to all volunteers via the mobile app, as specified in the System Objectives. On this page users can type new alerts into the input box provided, and send these out to users via the alerts page in the mobile app using the “Post Alert” button, as shown in *Figure 3.3.9*. This page will also contain a scrollable list of past alerts, for administrators to view or delete if necessary.	 Cases Alerts Add New Case Chat Alerts Type here to add new message... Post Alert Past Alerts SAVE THE DATE! Annual Charity Table quiz is taking place on the 21st April, all proceeds will be going straight into new equipment for the shelters. Hope you can all make it! Sun Apr 09 2023 12:12:22 GMT+0100 (Irish Standard Time) All volunteers please note that the bird flu is very common at this time of year, especially in robins. Always wear gloves! Sun Apr 09 2023 12:11:00 GMT+0100 (Irish Standard Time) *Figure 3.3.9 – Alerts*
Chat The Chat screen allows the administrators and volunteers to communicate with each other in real-time using a group chat messaging system. Messages sent are stored in the database and separated by sender ID. As shown in *Figure 3.3.10*, the page consists of a chat box, where users can type their messages, and a container displaying all of their messages as well as responses sent by others. This chat page is linked to both the web and the mobile applications so that administrators and volunteers alike can communicate effectively.	 Cases Alerts Add New Case Chat Type a message... Apr 9, 2023 Hi guys, what happened with that swan last week when you brought it to the shelter? Hi David, the people at the shelter said they should have him on the mend in no time! So all good there Thanks for checking in! Hey guys, just letting you all know I saved that fox in Trinity College earlier! Good to be back on campus, it's been in a while 😜 Good one Joe!! *Figure 3.3.10 – Chat Page*

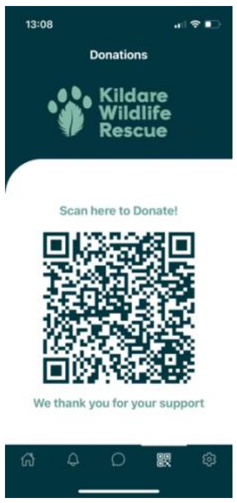
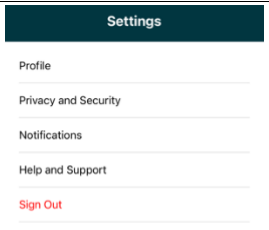
Mobile Application

Table 3.3.2 - Mobile App System Description & Screens

Description	***Image***
Login and Sign-Up Pages The login and sign up pages for the mobile application are shown in *Figures 3.3.11* and *3.3.12*. These are almost identical to the same pages in the web-application to maintain uniformity across platforms. Both functionality and layout of these pages is the same as the web-application above, however only users registered as “volunteers” will be able to log in, and any account created via the mobile app will consequentially be registered as a volunteer.	 13:08 Kildare Wildlife Rescue Log In Enter Email Enter Password Log In Don't have an account? Sign Up *Figure 3.3.11 - Login*  13:08 Kildare Wildlife Rescue Sign Up Enter your Full Name Enter Email Enter Password Upload Profile Picture Sign Up Already have an account? Login *Figure 3.3.12 - Sign Up*

Navigation Bar The Navigation bar is located at the bottom of every screen in this application, and allows the user to navigate to any one of five screens. The bar was given a simplistic and intuitive design, using icons to label each screen. Universally recognized icons were selected, as can be seen in *Figure 3.3.13*, to reduce confusion for the user and ensure usability.	 *Figure 3.3.13 – Mobile App Nav Bar*
Home Page The Home Screen, or landing screen, of the mobile application is where volunteers can view open cases of animals requiring rescue. These are displayed in a list format, filtered by closeness in location as shown in *Figure 3.3.14*. Users can tap on cases to open the Case Details page in order to view more details about the case and choose whether or not to respond. The home screen also contains a button to take the user to the Map View of cases, which is described below.	 13:08 Current Cases Phibsoboro Ferret Species: Ferret Status: Active Resp: None Dundrum Fox Species: Red Fox Status: Active Resp: None Raheny Robin Species: European Robin Status: In Progress Resp: Joe Bloggs *Figure 3.3.14 - Home Screen*
Map The map screen is similar in concept to the Home Page, but different in that it visualizes the cases on a map of your surrounding area. This screen makes use of the Google Maps API, and the icons dotted around the map indicate different locations where wildlife is in need of rescue. This page updates as cases are added to and removed from the database, with a real-time visualization of casefiles. Icons are also colour-coded to make it very distinguishable what the status of each case on the map is, and a key is provided as shown. Users can again tap on these cases to be brought to the Case Details page.	 13:08 Map View Active Cases In Progress Cases Dublin Google *Figure 3.3.15 - Map Screen*

Case Details The Case Details page provides the user with all of the relevant details about that case, including any media which has been attached. This page will display details of whichever case was used to navigate to it. The page also contains a button, which allows the user to respond to the case. If the user responds, that case status will change from Active to In Progress, and that user's name will be added to the responders. In some cases, there will be more than one responder to each case. If the user is already responding to the case, this button will change in appearance, as depicted in *Figure 3.3.18* across.	 13:09 Case Details Kildare Wildlife Rescue Additional Notes Fox appears to have a broken leg... Current Responders None Media *Figure 3.3.16 Case Details*  13:09 Case Details Kildare Wildlife Rescue Case Title Dundrum Fox Status Active Species Red Fox Member of Public - Name Joe Address 123 House Road *Figure 3.3.17 Case Details*  13:09 Case Details Kildare Wildlife Rescue Current Responders Joe Bloggs Media *Figure 3.3.18 Case Details*
Alerts The Alerts page in the mobile app, shown by *Figure 3.3.19*, is the landing zone for the alerts written by administrators in the web application. Here users will be able to read current and past alerts uploaded by the administrators, in real-time.	 13:08 Alerts **SAVE THE DATE! Annual Charity Table quiz is taking place on the 21st April, all proceeds will be going straight into new equipment for the shelters. Hope you can all make it!** Sun Apr 09 2023 12:12:22 GMT+0100 (Irish Standard Time) **All volunteers please note that the bird flu is very common at this time of year, especially in robins. Always wear gloves!** Sun Apr 09 2023 12:11:00 GMT+0100 (Irish Standard Time) *Figure 3.3.19* *Alerts Page*
Chat The Chat screen allows users to communicate with each other in real-time, similar to the chat screen in the web application. See *Table 3.3.1* for further details. Any messages sent by either volunteers or administrators will appear on this page.	 13:08 Chats Apr 9, 2023 Hi guys, what happened with that swan last week when you brought it to the shelter? 12:33 PM Hi David, the people at the shelter said they should have him on the mend in no time! So all good there 12:35 PM Thanks for checking in! 12:35 PM Hey guys, just letting you all know I saved that fox in Trinity College earlier! Good to be back on campus, it's been in a while 😆 12:56 PM Good one Joe!! 12:56 PM Type a message... *Figure 3.3.20* *Chats Page*

Donations Page The Donations page was a bonus requirement from the client, and is exhibited in *Figure 3.3.21*. This page is accessible from the Navigation Bar and simply displays a QR Code which, when scanned, leads to a donations page for the wildlife charity. When attending the scene of a rescue, volunteers will often meet the member of the public who reported the animal, and both in this scenario and many others it can be often necessary to quickly direct people to the relevant link for donating to the charity. Having the QR Code on hand ensures that this process is very straightforward and swift.	 13:08 Donations Kildare Wildlife Rescue Scan here to Donate! We thank you for your support *Figure 3.3.21* *Donations Page*
Settings The settings page displays the menu shown over, and allows the user to log out of the application. Functionality will be added to the other menu items at a future date.	 Settings Profile Privacy and Security Notifications Help and Support Sign Out *Figure 3.3.22* *Settings Page*

4 DESCRIPTION OF THE WORK DONE

This section of the report will provide an analysis of the work completed over the course of the project. It details the stages involved in the project, from requirements gathering and planning to development, implementation, and eventually, testing and evaluation. The methodology used for the project is explained, along with a list of software/tools used and the required operating environment for the software. It will begin referencing the definition of requirements, and then delves into the specific technical aspects of the project, including the database design and implementation, the chat messaging system development, and how the system fulfils the requirements of the client. This chapter aims to provide a comprehensive understanding of how the software was developed and how it works.

4.1 Definition of the System Requirements

Before commencing the design and development of the system, the client was engaged with regular meetings and emails in order to establish the basic requirements. With very little understanding of the inner-workings of a wildlife rescue charity, it was vital to determine why the app was necessary, and exactly what was expected to be delivered from the client side. As the project progressed and time passed, the client added several additional functionalities and requests to the initial requirements, extending the early scope of the project. The final requirements are detailed in *Section 3.3* as System Objectives. The overall goal of this project was to improve the efficiency of operations in the wildlife charity, by creating a centralized application through which users could view and respond to cases, and message each other in real-time via a group chat. The current process in place is outlined in *Section 1.2*.

Having no existing application or basis to work from resulted in a certain degree of freedom being granted by the client, as long as their basic requirements were met and the software beinused remained free. It is also worth mentioning that the client operates a web and app design firm in Dublin City Centre, so instruction was provided not to spend a large portion of time on the design of the system, but to focus instead on development and functionality, and that a more aesthetic design would be implemented at a later date.

Meetings were conducted both in-person and online, with regular emails being exchanged especially in the early period of the project, and also throughout. The entire system was being designed from scratch, with no previous technical infrastructures placing any restriction on development. The client did mention however, that there is a system in use in animal rehabilitation centres which may be useful to connect to in future. However, it was decided not to pursue this course of action as the client had little understanding of the inner workings of these centres, dealing himself only in rescue and not rehabilitation.

4.2 Choosing Suitable Technologies

Once the requirements for the project had been established, and it was clear that no technical restrictions applied, research was conducted into the various tools and software with which the system could be developed. This research aimed to identify appropriate solutions for the client-side, server-side, data storage, the chat system and source code editor and version control. The research evaluated potential solutions based on their applications, capabilities, compatibility and lastly whether or not the developer had experience with or knowledge of the solution, which in most scenarios was not the case.

With extremely limited developmental experience, the initial research conducted was into general and basic mobile and web application development. Having some degree of exposure to Google’s Flutter SDK and the Dart programming language for cross-platform application development, the initial idea was to employ the use of this SDK for the purpose of this project. However, the developer was eager to test themselves and employ the use of a different technology, in order to somewhat steepen the learning curve. Although, this may not have been the most practical approach, particularly from an efficiency perspective, the project met all the requirements nonetheless, and the developer has little regret in pursuing this course of action.

Front-End Application

The initial research was conducted into alternatives for the Flutter SDK. In preliminary discussions with the client, they expressed the necessity for the mobile application to run on both iOS and Android platforms, and hence a cross-platform software was sought after as this would remove the need for two separate applications, as is often the case. All research seemed to point towards React Native, which is Meta's UI software framework for developing applications for iOS, Android, Web and more. React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It's based on React, the JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms.. Although there were some alternatives, this technology met all of the requirements for the front-end of my application and appeared the most straightforward course of action. With a capable level of Java programming, it was decided that learning to use JavaScript and React would be an achievable task. Research also pointed to the versatility of this software, and its compatibility with many other frameworks and technologies, as well as its precedence in industry, and hence React Native was chosen for the Front-End development.

There are two ways to build a React Native application from scratch, either with the React Native CLI, or with the Expo CLI. Using the React Native CLI is a more complex and time-consuming method of development, especially for inexperienced developers (Medium, 2021), and hence it was decided that Expo was the optimal choice. Expo is a free and open-source platform used for building, deploying, and managing React Native applications. It simplifies the process for developers with less experience, while still allowing for powerful applications to be built (ExpoDev, 2023).

Using Expo for React Native development offers several benefits, including:

- ***Simplified Development***: Expo provides built-in libraries and components that make it much easier to develop React Native applications. It eliminated the need for the developer to write complex code and configure settings, which could have significantly slowed down the development process.
- ***Easy Deployment***: Expo also provides a straightforward process for deploying React Native applications to the app stores. In the case of this project, this was not relevant but was regarded as a benefit nonetheless.
- ***Cross-Platform Development***: Expo allows developers to build cross-platform React Native applications that can run on both iOS and Android devices, without requiring significant changes to the codebase. It also allows for instant deployment to a physical device, via the Expo Go application.

For this project, using Expo for React Native development was beneficial as it simplified the development process greatly, allowing the developer to focus on building the application's functionality rather than worrying about the underlying technical details.

Back-End Application

For the back-end of the system, various options were considered. The technologies reviewed can be separated into two categories, Mobile Backend-as-a-Service (MBaaS) and Custom

Backends. In general, it was determined that although they may be more flexible to the needs of the application, Custom Backends such as Django or Express.js were more complex, time-consuming and expensive to build and hence these were removed from consideration. MBaaS platforms such as Firebase, AWS Amplify and Back4App would allow the project to meet all of the requirements necessary without adding extensively to the complexity of the task. Firebase was chosen as the optimal platform, due to its compatibility with React Native, the accessibility to related learning materials, the fact that is widely used and proven in industry, and finally the fact that services such as authentication and push notifications are provided for free, which is not the case for some other platforms. While other platforms could potentially have been used and may have fulfilled the same purpose, there were no flaws seen in selecting Firebase as the back-end structure for the system.

Data Storage

Primary research indicated the requirement for a database to be designed for this application. Various SQL and No-SQL options were reviewed, and having prior experience with this language the developer was opting for an SQL database. However, after selecting Firebase as the back-end platform, it was realised that Firebase provide database management services, and a cloud-based data storage system. Hence, it was established that an external database would not be necessary.

Chat System

The chat system for this application has several basic requirements. Firstly, it is required that the app support sending and receiving of regular text messages, as well as photos and videos. It must also be capable of supporting a group chat function, as this is the purpose of the chat system in this application. Several APIs were reviewed in the initial research, such as SendBird and Twilio, however all of the options found only offered a 30-day free trial, after which point they charged a monthly subscription. This would not suffice, as the requirements stated that the project must be free of charge, and hence these APIs were disregarded. Having selected React Native and Firebase as my front and back-end platforms, I then began to research other applications which implemented these and supported chat systems. This unearthed the “react-native-gifted-chat” library, a complete chat UI for React Native. This library fit the requirements perfectly and was highly compatible with both React Native and Firebase, and hence it was the obvious selection.

Source-Code Editor

Visual Studio Code was selected as the source-code editor for this project. This tool proved extremely useful due to its ability to connect seamlessly to simulators to compile the app, and its built in Git support which aided in version control and management. Alternatives were considered such as Android Studio, Atom and more, however the developer also had previous experience with VS Code and hence it was the optimal choice.

Version Control

During the development process, version management was a crucial tool that enabled developers to create secure and safe branches of the system to test and develop new features while maintaining a working version in a separate branch. In addition, version management added an extra layer of security to the project, allowing the system to be reverted back to a previous iteration if necessary. For this project, GitHub was chosen as the repository management tool for several reasons, including: the developer's previous experience with the platform; its compatibility with VS Code; and its reliable hosting service, which ensured that all versions and branches of the solution were securely backed up.

4.3 Designing the System

After the requirements had been established and suitable technologies selected, the next phase of the development process began, the design phase. The design of the system's functional components and the client-side application's appearance were both part of this project phase. The following section outlines the steps taken to arrive at the final system design.

4.3.1 User Interaction Analysis

The first step in the system design involved an analysis of how users would interact with the system on a daily basis. This was an essential component of the design as it elucidates how the user will communicate with the database and traverse the application itself. In order to properly understand the user's interactions with the system, the following heuristic tools were used:

- Scenarios
- Hierarchical Task Analysis (HTA)

To analyse a specific user's interaction with the system, a series of case-based scenarios were created. These scenarios provided a linear narrative of the user's interaction with the system without any special conditions or branching. Multiple scenarios were developed to gain a better understanding of the user's requirements and interactions. The scenarios were used to develop the user's activity flow, and in conjunction with hierarchical task analysis (HTA), the steps and tasks involved in achieving specific goals within the system were textually broken down. HTA allows for more detailed analysis of tasks, including branching tasks and special conditions, and can help identify redundant steps and improve system efficiency. By using both scenarios and HTA, the designer can gain a more comprehensive understanding of the user journey and optimize the system accordingly. The resultant user interactions can be visualised through the flow diagrams found in Appendix C.

4.3.2 Web Application Design

With regard to the web-based application for administrators, it was imperative to the project that this system would be intuitively constructed, with little room for human error, and that it be easily traversable and simplistic in its design. The final layout for the web-application was arrived at after a series of steps had been taken, including planning, low-fidelity prototyping and high fidelity prototyping.

4.3.3 Mobile Application Design

Similar to the web application, the mobile app necessitated an extremely simplistic and intuitive design, and many steps were taken to ensure that this was the case. These steps are outlined in greater detail below.

4.3.4 Planning the Design

In order to expedite the design phase, a brief brainstorming session was conducted in which a series of colour palettes and theme ideas were gathered for both the web app and the mobile app. Inspiration was taken from the Kildare Wildlife Rescue logo, which is displayed on the Login and Sign Up pages, as well as the top right corner of the Home Page. The darkish turquoise colour from this logo is found consistently throughout both applications, in order to draw the user's attention to certain components such as Submit buttons. This aids in making the system easily traversable by promoting user recall and maintaining consistent design throughout the apps.

4.3.5 Low Fidelity Prototyping

Once a general idea was in place of how the application was going to be laid out, low-fidelity prototyping was the first step in transforming these ideas to physical concepts. This involved producing hand-drawn sketches of the critical screens of the application, which aided in visualizing the application and generating component templates. These templates were then evaluated, edited, and improved before being applied across other screens to create a consistent application structure. The prototypes primarily focused on the screen structure and drew inspiration from the scenarios analysis and Hierarchical Task Analysis (HTA) in their design. Following this, the HTA was reviewed and revised in accordance with the prototypes, following an incremental development approach.

4.3.6 High Fidelity Prototyping

After completing the low fidelity prototypes, these were transformed into high fidelity prototypes using software. For this task, Figma was utilized to visualize the screens. Similar to the low fidelity prototypes, the high fidelity prototypes emphasized the screen structure and activity flow. However, in this iteration, other design aspects, such as component, shape, size, and colour, were the focus. This was done to ensure that the design was clear, consistent, and aesthetically pleasing, thus enhancing the usability and accessibility of the system. A consistent design for forms, buttons, and other components was established, and this was then replicated throughout the application to promote user recall and usability.

4.3.7 Chat System Design

In order to be fully prepared to begin development of the system, the chat system was also designed in order to figure out how the messaging system was going to work. Through further liaison with the client, it was determined that only a forum-style group chat would be necessary for the charity, as private chats can be and in most scenarios will be, carried out via standard messaging apps i.e. WhatsApp. Hence, the design for the chat system was slightly simplified, as all messages could hence be stored in one table in the database, as long as the sender ID was attached to each message.

4.4 Developing the System

The developmental phase of the project commenced when all designs had been finalised. This began with installation of the necessary packages and software, after which the front-end of the application was created. The mobile application was developed first, as this was seen as bearing the steeper learning curve, and this was followed by development of the web application.

4.4.1 Installation of Necessary Software

The first step in installing the necessary software was to install Node, a server-side JavaScript runtime environment. This software is what will build the JavaScript code, specifically using a server-side build script called Metro. This installation included the Node Package Manager (NPM) and Node Package Executor (NPX), which are vital tools in the development of applications. Next, XCode was installed on the developer's local machine, in order to allow for iOS development and testing on simulators. Android Studio was also installed to make use of the in-built Android emulators. Following this, the Expo CLI was installed using the NPM. Expo is an open-source framework for apps that run natively on Android, iOS and the web. It allows for apps to be built quickly and easily using a React

Native code-base. After these had all been installed, a basic app was created and the developmental work commenced.

4.4.2 Developing the Back-End

Before the front-end application could be properly configured, the back-end of the system needed to be formed. As mentioned previously, the back-end for this system was created using Firebase, Google's Mobile Back-end As A Service product. Firebase allows developers to build applications quickly and easily, with the knowledge that their data is secure and their system works. It also provides extensive documentation with tutorials to make it the optimal back-end provider for inexperienced developers. The services provided by Firebase include Authentication, Database management, File Storage and much more. Firebase also connects to React Native applications seamlessly and straightforwardly, again making it an ideal choice for the inexperienced developer. Below are some descriptions of the different features implemented in this system using Firebase.

Authentication

Firebase allows for Authentication to be made very simple for developers. Most applications need to know the identity of their end user, and in the case of this system that is definitely true. Firebase Authentication allows for a custom UI to be developed, and once a user enters their details, these are stored in the back-end and a user account is developed. Once a pre-created user account enters their login details into the login page, three events take place. Information about that user is returned to the device via call-backs, allowing the developer to personalize the user experience for each specific user. A unique user ID is also passed, which decides the types of data that will be visible for that user from the back-end system. A log is also maintained automatically of the user session, which allows for users to remain logged into the system even after closing the application or browser. All of these features were beneficial to this system, as they both simplified the development process and improved the user experience with the two applications.

Overall, Firebase Authentication was very useful for this project. One slight downfall is the inability to authenticate users with different permissions and roles, however this was overcome by storing the users in the Cloud Firestore, as explained below.

Cloud Firestore Database

The Cloud Firestore Database feature allows for any and all application data to be stored in the cloud. For this system, the use of a database was essential and Cloud Firestore was the optimal choice. It is a cloud-hosted, NoSQL database which can be accessed directly by both the mobile and web applications using native SDKs. The database operates by storing data as documents, which contain fields mapping to values. The documents support a wide variety of data types, from simple strings and numbers to complex, nested objects. Documents are stored inside collections, which separates the data similar to tables in a more common database management system. Firestore also allows for subcollections, which grants the developer the opportunity to build hierarchical data structures which can scale as the application grows.

For this system, Cloud Firestore was utilized for four main reasons, to maintain records of users in a hierarchical manner, to store messages from the group chat system, to store the alerts posted by administrators for all volunteers to read, and arguably most importantly, to store the details of each individual case of an animal requiring rescue. The data maintained in each of these collections is outlined below.

1. *Users Collection*

Table 4.4.1 - Users collection description - Firestore Database

Fields	***Sample Entry***
Display Name	Joe Bloggs
Email Address	joebloggs@gmail.com
User Type	Admin
User ID	3K10zWQ25HJHl4LCoJvsf2
Profile Photo URL	https://i.pinimg.com/1f8c1ecbdaf2d43.jpg

This user data is stored as a result of one flaw in the Firebase system. Different user types cannot be created using Firebase Authentication, hence all user data must be duplicated in the Cloud Firestore so that the 'type' field can be added, which specifies in this system whether the user is an administrator or a volunteer. Only administrators should be able to access the web application, hence it was necessary to maintain details of each user in the database. User IDs are unique to each user, and are created by Firebase Authentication when the user account is created. The profile photo URL field stores a link to a Firebase Storage file, which will be discussed in the next section of this report.

2. *Messages Collection*

Table 4.4.2 - Messages collection description - Firestore Database

Fields Maintained		***Sample Entry***
Message ID		Bk3gbhjhvGH12Jbj3ek
Created At		March 1, 2023 at 12:06:20PM UTC
Text		"Hello, how are you"
User	**ID**	3K10zWQ25HJHl4LCoJvsf2
	Photo URL	https://i.pinimg.com/1f8c1ecbdaf2d43.jpg

Messages from the group chat are stored in a collection such as the one above, with the message again being unique to each message and created when the message is sent. The 'Created At' field maintains the date and exact time at which the message was sent so that chats can be filtered and ordered chronologically. The text field is of course the message that has been typed by the user. Lastly, the User field is a subcollection containing the details of the sender. This subcollection does not require all of the details of the sender, only their ID and the URL of their profile picture so that this can be displayed beside the message bubble.

3. *Alerts Collection*

Table 4.4.3 - Alerts collection description - Firestore Database

Fields Maintained	***Sample Entry***
Alert ID	Vfau76fIFqy8wghbqf2
Created At	March 1, 2023 at 12:06:20PM UTC
Text	“Volunteers, beware that Bird Flu is …”

Alerts are stored in a similar way to messages, except that the sender ID is in this case not necessary. Each alert is posted by an administrator for all volunteers to read, hence the identity of the sender is immaterial and will therefore not be stored.

4. *Cases Collection*

Table 4.4.4 - Cases collection description - Firestore Database

Fields	***Sample Entry***
Status	Active
Title	Dundrum Fox
Case ID	Kvjtfr7u6ifuFU6FKUuv
Species	Red Fox
Latitude	53.343005
Longitude	-6.256010
Member of Public Name	Joe Bloggs
Member of Public Address	3 Lane Road, Dublin

Member of Public Phone	087 1234567
Notes	“Fox appears to have broken leg…”
Files	Array of Photo/Video URLs
Responders	Jack, Jill
Created At	March 1, 2023 at 01:05:22PM UTC

This collection is perhaps the most important to the system, as it stores details regarding the animals requiring rescue. To summarise the order of events, a member of the public will call the telephone line for the wildlife rescue charity and speak to one of the administrators, providing details about the animal in question. The administrator will then make a record of each of these details and log them in the system. The status field will generally begin as ‘Active’ while the animal is still in need of rescue, this will then hopefully change to ‘In Progress’ when a volunteer responds to the case, and eventually to ‘Rescued’ when the animal has been tended to. The title of the case relates to a system that the charity use to refer to different cases, for example above is the ‘Dundrum Fox’. This is a quick way for volunteers and administrators to refer to cases within the group chat and by word of mouth as it triggers a recall much better than, say, the Case ID. Case IDs are again generated by Firebase and are unique to each case. The species of the animal is then recorded if known, as well as the exact location by longitude and latitude. The client expressed the need for exact locations, citing that in towns and built up areas a rough address would suffice, however in rural areas this is often not the case and a pinpoint location is necessary. The contact details for the Member of Public are then recorded, specifically their name, address and telephone number. Notes are included by the administrator to give any specific details that the volunteer may need to be aware of, and lastly photos and videos are included if applicable. Similar to the profile pictures, the media files are stored as URLs for the Firebase Storage files, explained below.

Overall the Cloud Firestore database system was very efficient in storing all of these different types of data. One slight drawback with this platform is that it does not support overly complex querying, as it is not an SQL database or even a relational database, however this was not an issue for this system as the data being stored is quite simplistic. Once Firebase had been fully configured and set up, the next step was developing the front end of the application and connecting it to Firebase.

Firebase Storage

Firebase Storage is a feature that allows developers to upload their application users’ files to the Cloud. For the purpose of this system, Firebase Storage’s main function is to store media such as photos and videos, both for rescue cases and simply for user profile pictures. Firebase Storage gives applications the capability to take files from the user’s device, upload them to the cloud and provide a reference to that upload, in the form of a URL. This URL can then be used throughout the application to display the media, whether it be in the chat screen as someone’s profile picture, or on individual cases to show the volunteers an animal’s whereabouts.

One major benefit in using Firebase Storage is that it eliminates the need for images to be saved onto a user's physical device, lessening the application's storage requirements. Images are instead saved in the Cloud and accessible by the download URL. This does require that the user have an internet connection, however this was an inevitability of this system, as the requirements for a real-time chat system and database mean that internet connection would always be necessary.

4.4.3 Developing the Front-End – Mobile Application

Development of the front-end application commenced and proved to be the most challenging step in the developmental process. Becoming familiar with React Native having no previous experience with React or even JavaScript posed a challenge. Despite this however, two fully-functioning applications were developed. The first step was creating a multi-screen application, with navigation between screens being intuitive and straightforward.

Navigation

Navigation in the mobile application was implemented using a Navigation bar, located at the bottom of the screen. This was created with a simplistic and intuitive design, to improve the usability of the application. Five main screens can be accessed from the navigation bar, namely the Home Screen, Alerts, Chats, Donations Screen and Settings.

Standard Screens

The standard screens (such as login and sign up, homepage etc) were developed first, with an uncomplicated design. The forms on these pages were created using simple 'Text Input' React Native components, and the Buttons using 'Button' components. Styling was added using the 'Style Sheet' components, which allows developers to change the appearance of the views and components that make up the app with ease. Buttons were styled in a dark colour so that they would stand out to the user, informing them indirectly that these are buttons to be pressed if needed. Placeholders were also used inside the Text Input boxes, to advise the user of the values required in each box. Each of these features were added to increase usability and to prevent erroneous behaviour by the user.

Chat Page

The chat page was added next, employing the use of the Gifted Chat library. Gifted Chat automatically creates the messaging screen, adds the input box at the bottom where the user can type their messages and renders the messages in the standard chat format on screen.

Map Screen

The Map screen was created using the react-native-maps library. This library allows the device to render a map natively on the device, using Apple Maps or Google Maps API. For Google Maps additional setup was necessary, as this API requires the developer to register a Google Cloud API project and create a unique Google Cloud API key. Once this was completed, the map was ready for use. Cases were located on the map using their longitude and latitude attributes, with which every case is uploaded. This allowed for icons to be placed on the map, highlighting the location of each case. The purpose of this screen was for volunteers to be able to see cases in their locality or close to their current location, in order to enhance the likelihood of a response. Icons were also colour-coded by case status, red for Active cases and Yellow for cases in progress. This allowed for the user to distinguish not only whether or not there are cases nearby, but also what type of case it may be.

4.4.4 Developing the Front-End – Web Application

Very similar to the mobile application, developing the web application proved itself quite a challenging task, due to lack of experience in JavaScript frameworks and programming. Development began with creating the various pages using React Native components. In general, the pages of this application were designed in much the same way as the Mobile Application above. Due to the fact that this webpage is only for internal use by charity administrators, however, focus was more on functionality and less on aesthetics. Login and Sign Up pages mirrored the mobile application in layout, as did the chat page, for uniformity and consistency purposes.

<u>Homepage</u>

The Homepage was designed using three ‘FlatList’ components, which allows data to be displayed uniformly. This allows for the three different types of cases to be displayed side by side. Here the user can view all case details, and should they wish to alter the details of a given case, this can be done by simply clicking on that case and changing the details in the pop-up that appears. This allows for case details to be updated with ease.

4.4.5 Connecting the Front and Back-End

Connecting the React Native applications to the Firebase project was a relatively straightforward process. It involved firstly adding the applications to the Firebase project on the Firebase Dashboard, and this then provided the Firebase SDK to be added to the React Native file structure. The firebase configuration credentials were added to each of the applications’ file stacks, which contained the API keys, app IDs and other vital details. Security precautions were taken with regard to these credentials, as is outlined in the next section of this report. Once the Firebase configuration file was fully set up, the Authentication, Database and Storage libraries were exported for use throughout the applications. Through a series of Firebase SDKs and libraries, all of the services that Firebase had to offer were at the developer’s disposal.

4.4.6 Security

In order to guarantee the system's security against both malicious attacks and erroneous user activity, several precautions were implemented. This section of the report outlines the steps and measures taken to achieve this objective. The security of the system was implemented through four main methodologies: Password security, Route Protection, Database Credential security and lastly Firebase security rules.

<u>Password Security</u>

Password security was implemented to maintain the security and privacy of user accounts. Weak passwords make it easier for hackers and malicious software to access user data (KeeperSecurity, 2022). According to Microsoft Security, a strong password is defined as having at least 8-12 characters, containing at least one uppercase letter and a combination of letters, numbers and symbols (Microsoft, 2023). Hence, user passwords for this system are required to have the same characteristics.

<u>Route Protection</u>

One of the critical requirements for any application that handles sensitive data is ensuring the security of routes to prevent unauthorized access. In web-based applications, route security is generally implemented using various techniques such as token-based authentication, session management, and role-based access control (RBAC) systems (Manar H. Alalfi, 2012). In mobile applications, route security is typically enforced by using a combination of session management and token-based authentication (Ramadan Abdunabi, 2013). For the purpose of this system, Firebase Authentication was used to secure the routes in both the web and mobile applications. Firebase Authentication provides a straightforward and secure way of authenticating users and managing their sessions. By requiring that the user is currently logged into the app, we were able to ensure that only authorized users could access the protected routes, i.e. the main body of the app. Users who are not logged in can only access either the Sign Up or Login pages. The code which implements this can be found in Appendix E.

In the case of the web application, further security was added to the routes by ensuring that the user was logged in, but also of type 'Admin'. This was to ensure that only the administrators could access the web app.

Database Credential Security

In order to connect to the Firebase backend a series of credentials are required, including an API key, application ID, database URL and more. These credentials are sensitive information, as anyone with access can therefore access the database, read and write to it, and potentially incur large costs. As a result, it is necessary to store these credentials as environmental variables to ensure the security of the database (Huls, 2021). The credentials were added as values to a *.env* file in the file stack, and subsequently to the *app.config.js* file under *extra.* This allows the variables to be called inside the Firebase configuration file without having to include the values themselves. When the applications were uploaded to GitHub, the *.env* file was omitted from both uploads in order to maintain the security of these values.

Firebase Security Rules

Firebase security rules are a set of declarative rules written within the Firebase console that determine the level of access that authenticated users have to Firebase resources. In this project, Firebase security rules were used to restrict access to sensitive data, such as user information and chat messages. By imposing rules on the Firebase database, the application can ensure that only authorized users are able to access certain data, while preventing unauthorized access and ensuring the privacy of users. The rules were created using the Firebase Security Rules language, which allows control over the types of access granted to users. By utilizing Firebase security rules, the application can ensure that data is protected and access is restricted appropriately. This also prevents hackers or malicious software from accessing the data stored in the database and even causing price increases.

4.5 Testing the System

To ensure that the system was fully functioning and operating as envisioned, rigorous testing was carried out on the system throughout the developmental process. Testing was an integral part of the development of this system, as it uncovered any flaws in the design. There were three key testing methods implemented throughout the course of this project, in Unit testing, System testing and Client/User testing. Each of these methods are outlined below.

Unit Testing

Unit testing is a software testing technique in which individual units or components of software are tested in isolation from the rest of the system in order to ensure that they function correctly. The use of Unit testing helped to catch bugs early in the development process, before they could affect other parts of the system. Unit testing was performed using an automated testing frameworks called JestJS, which is commonly used with React Native. By writing unit tests, the developer increased the reliability and maintainability of the application. Functions such as *toBe*, *not.toBeNull* and *toMatch* were used to validate system outputs with expected results in order to identify edge cases and issues with functions, and solve these directly. Descriptions of individual tests are outlined in Appendix F.

System Testing

System testing was carried out to ascertain whether or not the individual components of the system are behaving and interacting with each other in the way that they are expected to. Also known as integration testing, these tests were carried out using a range of different methods as outlined here:

Table 4.5.1 - System Testing Methodologies

Method	***Description***
Protected Route Testing	Tests were carried out to ensure that users who were not logged in as verified users could not access any of the protected routes within the applications, and that only administrators could access the web application
Form Input Validation Testing	Tests were conducted to verify that sufficient error messages were displayed when invalid entries were made into forms within the apps. For example, if setting a password and the entry did not meet the requirements, or if a case was given an invalid status etc.
Account Creation	Test accounts were created to ensure that the account registration was functioning appropriately. Both Volunteer and Administrator accounts were created for this.
Data Flow Testing	Tests were carried out to verify that data entered into forms throughout the applications was being handled in the appropriate manner and as expected. In particular, the chat system was tested thoroughly to ensure that messages were being stored and displayed correctly. Cases were also tested rigorously to verify the same. Lastly, it was validated that photos and videos which were uploaded to the cloud storage were handled properly also

Client/User Testing

For the purpose of better understanding the end user, the client was asked to use the applications during the course of selected meetings. This allowed for more accurate information on how the user will interact with the system, which provided advice on usability and accessibility. It also allowed the client to provide feedback and what they thought could be improved in the applications, or what aspects should potentially be replicated elsewhere. In many ways this was the most useful testing method, as it ensured that the client's needs and preferences were sufficiently considered.

4.6 Difficulties Encountered

Throughout the course of this project, a wide array of varying difficulties were encountered. Undertaking a project of this magnitude with little to no experience in application development can be an extremely challenging and daunting task. Some of the specific difficulties encountered are described in more detail below.

Using New Programming Languages and Software

Having no previous experience with React Native, React, JavaScript or any web-design software proved to be a significant hindrance for the duration of the project. With minimal experience using Flutter for mobile app development, there were some similarities which could be transferred, however in general most of the material was learned from scratch. Substantial time was allocated to learning and becoming more efficient with JavaScript in the early stages of the project. Several resources and learning materials were utilized to acquire knowledge of the main aspects of the language, such as the syntax, the data types and structures, the libraries and frameworks which can be implemented. Once the developer had become somewhat comfortable with JavaScript as a language, the next task was understanding React, in particular the JSX syntax, React hooks, state and props. These elements of the language were most challenging to understand, and again various resources were employed in the process of learning this framework. Lastly, React Native was introduced, and with only some slight differences to React, this was the most straightforward phase of the learning cycle. A separate task was learning to use Firebase as a backend, and again in this case the developer had no prior experience of using this platform or anything similar. Lastly, with no knowledge of properly testing a complete application, the developer encountered difficulty with this phase of the development cycle, specifically with unit testing.

Choosing the Tools and Frameworks

The developer also faced challenges in selecting the software and frameworks with which to develop the applications. With no prior experience it was a complicated problem, weighing up the pros and cons of each technology without any experience in using them. While several resources could be utilized to learn about each technology, most were slow to draw attention to the flaws in each software. This meant that the developer was unsure of which frameworks would be the optimal choice and created certain levels of doubt throughout the entire project.

Overcoming the Difficulties

Difficult as it may have been, all of these challenges were thankfully overcome. The primary solution to most difficulties faced was hard work and perseverance, and when combined with time no problem was insurmountable. Several resources were also utilized, including online learning platforms such as LinkedIn learning and Udemy courses, books, articles and instructional videos on platforms such as YouTube and CodeAcademy. Lastly, several open-source code bases were accessed via GitHub in order to aid the development of these applications. These aspects, along with the help of my tutor and lecturers allowed the knowledge that was lacking to be acquired, and the challenges to be overcome.

5 CONCLUSIONS AND RECOMMENDATIONS

This section of the report will outline the conclusions that were reached over the course of this project. It also provides recommendations for future development and enhancement of this system.

5.1 Conclusions

In conclusion, the development of the mobile and web-based applications for the wildlife charity has been a challenging yet rewarding experience. The objective of the project was to provide a platform for the charity to efficiently manage and respond to animal rescue cases, as well as to allow for more effective communication between administrators and volunteers in the charity. The applications necessitated a positive user experience through its usability and functionality. The final product has satisfied all of the Terms of Reference of the project. The complete applications have been developed using the JavaScript, JSX, NodeJS, React Native, Expo CLI and Firebase, with both front-end and back-end services. Rigorous testing was carried out using a variety of methods, as was described previously in the report, to ensure that each component of both applications is functioning as expected and required. The necessary security precautions have also been taken to ensure the safety and integrity of the application and itself and that data stored in the database. And lastly, the system can be operated completely free of charge, which was an important requirement for this project.

Despite the challenges faced during the development process, such as limited experience with application development and time constraints, the final product has met all of the project's, and the client’s, objectives and requirements. The success of this project has demonstrated the importance of using the right tools and methodologies for application development, as well as the significance of effective communication and collaboration between the developer and the client.

5.2 Recommendations for Future Work

While it has been said previously that the project has met all of requirements and terms of references specified by the client, and can therefore be branded a success, it is worth mentioning some areas where further development or improvement may take place.

- <u>Chat System:</u> For the purpose of this project, a singular, forum-style group chat was implemented to cater for text conversation between administrators and volunteers in the charity. However, in future it may be worth expanding on this component and creating a more dynamic messaging system, allowing for private messaging between users and creation of smaller group chats.
- <u>Connection to existing Wildlife Platforms:</u> The client has alluded to existing applications and software in rehabilitation centres which would be worth researching. If it were possible to sync these applications with those developed for the purpose of this project, it would allow for one centralized system for the rescue, rehabilitation and post-release monitoring of animals in our eco-system. This would be a hugely beneficial tool for wildlife organisations and for wildlife conservation in general.
- <u>Case Assignment:</u> One potential point for future development within the system would be to cater for a case assignment functionality, which would allow administrators to assign cases to users who may be most suited to that particular case, or if the more complex cases are to be pre-assigned to particular users/ groups of users.
- <u>Statistical Analysis:</u> As there was no historical database to reference, there was little statistical analysis that could be conducted of real cases. In future after the system has been in operation for some time, it may be possible to conduct proper statistical analyses based on real data.

TRINITY COLLEGE DUBLIN

Management Science and Information Systems Studies Project Report

Wildlife Rescue Management System: Developing Mobile and Web-based Applications for a Charitable Organization

APPENDICES

April 2023

Prepared by: Jack Cleary   Supervisor: Alessio Benavoli

APPENDIIX A – ORIGINAL PROJECT OUTLINE

Client:	Pearse Stokes – Wildlife Rescue Charities
Project Title:	Wildlife Rescue Management System: A Mobile and Web-Based Application for a Wildlife Charity
Client Contact:	Pearse Stokes
Project Supervisor:	Alessio Benavoli

Client background

Wonder Works is a design agency based in Dublin run by brother and sister Pearse and Dara Stokes. We have worked on the design side of a range of apps, including the world's first Augmented Reality Tourism App (with Failte Ireland) and various transport apps for the NTA / TFI.

Pearse has been involved in Wildlife Rescue for 7 years. Over that time he has transformed many of the processes and protocols used in Wildlife Rescue with a view to professionalizing the sector. He hosts the Wildlife Rescue Podcast and created a company called We Help Wildlife that aims to support independent Wildlife Rescues through training, fundraising support, etc.

Project background and Requirement

We are looking to develop an App that Wildlife Rescues can download, have a management side and a volunteer side. This App could replace the many WhatsApp group chats (which were meant to be replaced by Slack but that gained no traction).

The App would benefit if through its chat function we could have something like a 'casefile'. A number, associated photos/videos and exact location. This would enable the management to issue a new rescue case to the group chat, and for a volunteer to take that case. The case could be passed along and marked closed. All data would feed to a database for management review (and this data is crucial for ecological surveys and Wildlife Rehabilitation licensing).

Years ago I looked at using a sports team app for this, as it had similar functions, and perhaps there's even Human Rescue / First Response apps in existence that could serve as a guide. I'd love to chat to someone who knows about this stuff! There are apps for managing animal shelters.

What is involved for the student?

Scoping, the overall design of the information flow, programming and testing the app, etc.

APPENDIX B - INTERIM REPORT

Project:	We Help Wildlife - Mobile Application Development
Client:	Pearse – Kildare Wildlife Rescue
Student:	Jack Cleary - *19333982*
Supervisor:	Alessio Benavoli

Review of Background and Work to Date

The client has been involved in Wildlife Rescue Organisations for 7 years, in recent times primarily with Kildare Wildlife Rescue. The client requires a mobile application which would generally improve the overall system in place at the moment in many wildlife charities across the country. As of now, all communication is carried out via numerous WhatsApp group chats, and is overall very inefficient.

A mobile application could allow for all communication to be carried out in a centralized format. The client wishes for users to be able to view active rescue cases in-app, and respond to them if they choose. They should also be able to communicate via a chat function, similar to Slack, separated into various channels as required. From an administrative point of view, casefile data should feed to a database for review and analysis. Thus far, I have:

- Met with client and gathered all requirements, communicating effectively in order to properly understand the client's needs and wishes.
- Researched various software's that I could potentially use for developing this app and have settled on React Native as the optimal choice.
- Installed all necessary software's and began teaching myself to use React.
- Researched backend system structures which will be required for this project, and taught myself about system design using a number of different resources.
- Reviewed some open-source projects with somewhat similar scope to look for relevant ideas.

Terms of Reference

- Design and implement a mobile application for volunteers to improve efficiency, which will allow admin users to upload case files and volunteer users to select cases for responding to.
- Develop a chat function in the app to allow for proper communication.
- Potentially create a user manual outlining how the application works and how it can be applied to a wildlife rescue charities, if this is not extremely intuitive.

Further Work

<u>December & Vacation:</u>

- Continue my research into the backend system design that will be required for this project.
- Begin designing the front end using React Native and explore possibilities using open-source projects with similar scope.
- Begin backend design.

<u>Semester 2 – Weeks 1-7:</u>

- Complete development of the application.
- Consult client and tutor to look for areas of improvement/changes required.
- Implement any changes necessary.

Semester 2 – Weeks 8-11:

- Write and submit final report.

Conclusions

A plan has been put in place to allow me to fulfil the terms of reference for this project. The project will involve creating a user-friendly mobile application for wildlife rescue organisations to run efficiently and smoothly. It should be noted that due to the complex nature of this project, it is possible that difficulties and unforeseen obstacles will be encountered during its completion. However, with regular communication with my client and tutor, and a base knowledge of application development, I am confident that a practical and user-friendly mobile app will be produced in the timeframe.

APPENDIX C - DESIGN DOCUMENTATION

This section of the Appendix details the process by which this system was developed, including the development methodology employed, coding standards and conventions maintained, the flow of data and the database structure.

C.1 - Development Methodology

A Software Development methodology is a framework that is used to aid developers in the planning and organization of a software development project. This project was completed using an incremental development model, which is a methodology based on the concept of developing an initial prototype and continuing to develop and evaluate it throughout a number of iterations (Powell-Morse, 2016). The purpose of choosing this model was to allow for changing requirements over the course of the project, as this would allow for client feedback and input to be addressed without causing any negative repercussions for the project plan. The diagram in *Figure C.1.1* provides further detail to explain this methodology.

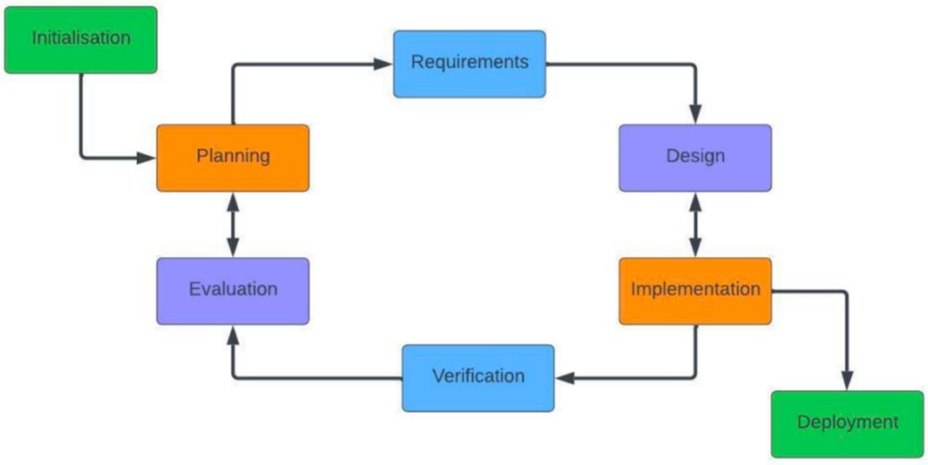


Figure C.1.1 - Incremental development model

C.2 - Coding Standards and Conventions

Coding conventions are a set of guidelines which maintain code quality, consistency and readability (W3Schools, n.d.). The following conventions and standards were maintained throughout this project:

- <u>Naming Conventions:</u> Meaningful and descriptive names were given to variables, functions and components. lowerCamelCase was used for variables and functions, e.g. “variableName”, and UpperCamelCase was used for naming components, e.g. “ComponentName”.
- <u>Indentation:</u> Loops and statements were 2 space indented.
- <u>Comments:</u> Code was sufficiently commented wherever necessary to make the code more easily understandable for future developers.
- <u>Code Structure:</u> Code was structured consistently and clearly, with files separated into folders for better organisation.
- <u>Error Handling:</u> Proper error handling techniques were employed to handle unexpected errors or erroneous behaviour by the user.

C.3 - Data Flow Diagrams

The following diagrams depict the flow of data in various scenarios, both throughout the system’s front end and back end, and to and from the database.

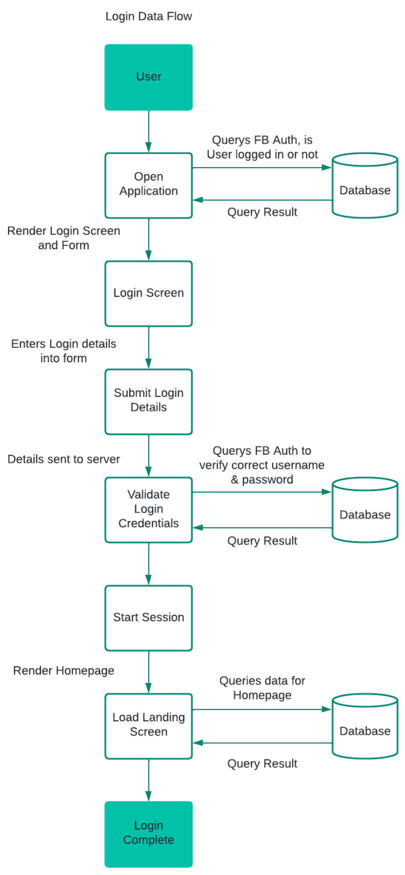


Figure C.3.1 - Logging In Dataflow

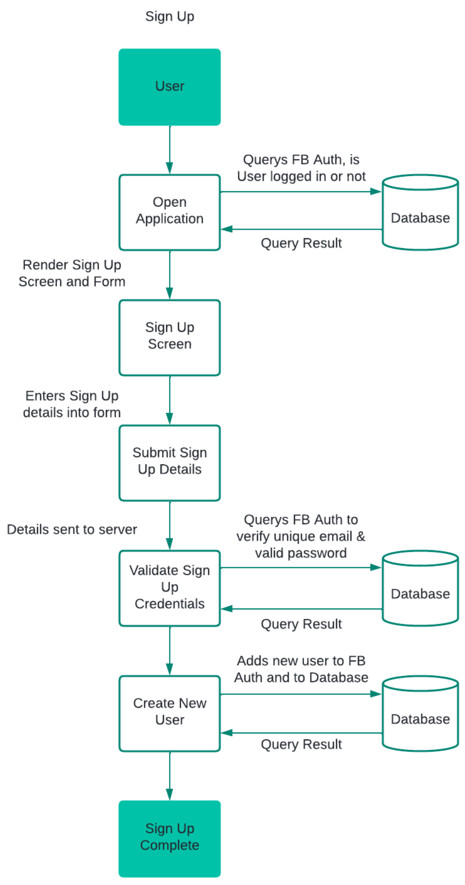

Sign Up
User
Querys FB Auth, is User logged in or not
Open Application
Database
Query Result
Render Sign Up Screen and Form
Sign Up Screen
Enters Sign Up details into form
Submit Sign Up Details
Details sent to server
Querys FB Auth to verify unique email & valid password
Validate Sign Up Credentials
Database
Query Result
Adds new user to FB Auth and to Database
Create New User
Database
Query Result
Sign Up Complete

Figure C.3.2 - Signing Up Dataflow

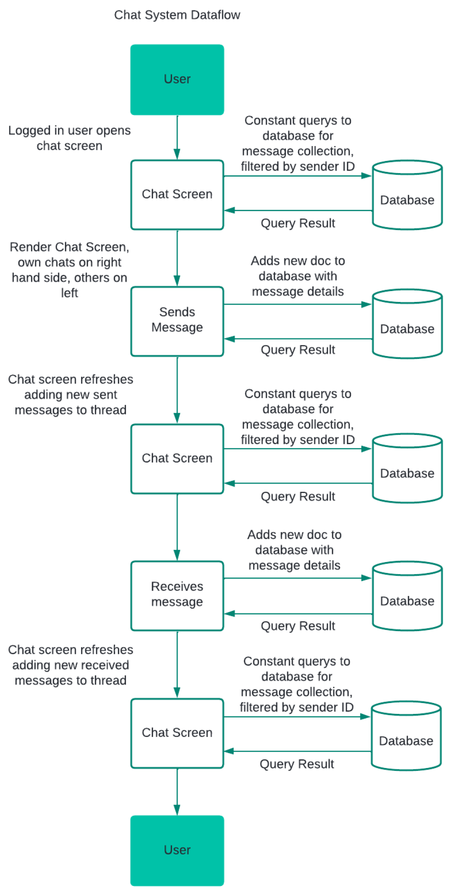

Chat System Dataflow
User
Logged in user opens chat screen
Chat Screen
Constant querys to database for message collection, filtered by sender ID
Database
Query Result
Render Chat Screen, own chats on right hand side, others on left
Sends Message
Adds new doc to database with message details
Database
Query Result
Chat screen refreshes adding new sent messages to thread
Chat Screen
Constant querys to database for message collection, filtered by sender ID
Database
Query Result
Receives message
Adds new doc to database with message details
Database
Query Result
Chat screen refreshes adding new received messages to thread
Chat Screen
Constant querys to database for message collection, filtered by sender ID
Database
Query Result
User

Figure C.3.3 - Chat System Dataflow

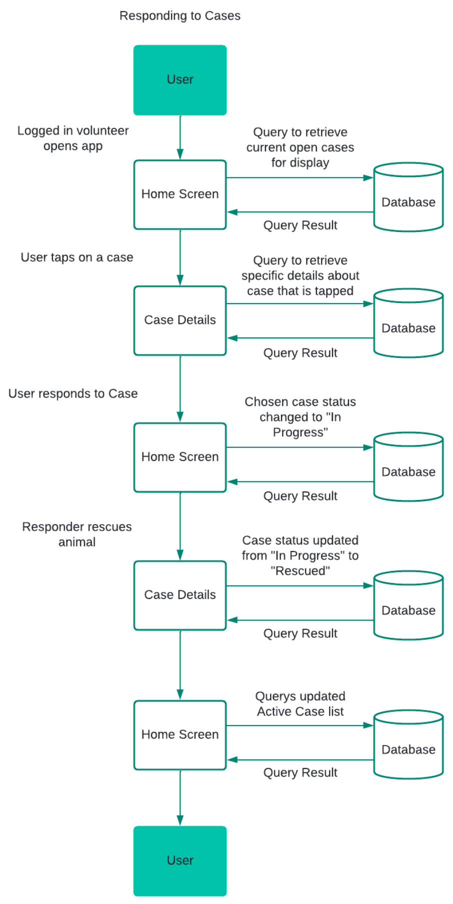

Responding to Cases
User
Logged in volunteer opens app
Home Screen
Query to retrieve current open cases for display
Database
Query Result
User taps on a case
Case Details
Query to retrieve specific details about case that is tapped
Database
Query Result
User responds to Case
Home Screen
Chosen case status changed to "In Progress"
Database
Query Result
Responder rescues animal
Case Details
Case status updated from "In Progress" to "Rescued"
Database
Query Result
Home Screen
Querys updated Active Case list
Database
Query Result
User

Figure C.3.4 - Responding to Cases Dataflow

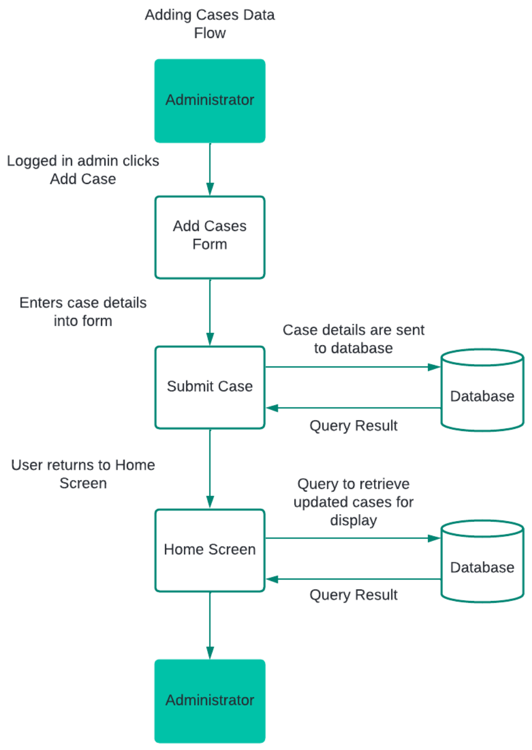

Adding Cases Data Flow
Administrator
Logged in admin clicks Add Case
Add Cases Form
Enters case details into form
Submit Case
Case details are sent to database
Database
Query Result
User returns to Home Screen
Home Screen
Query to retrieve updated cases for display
Database
Query Result
Administrator

Figure C.3.5 - Adding Cases Dataflow

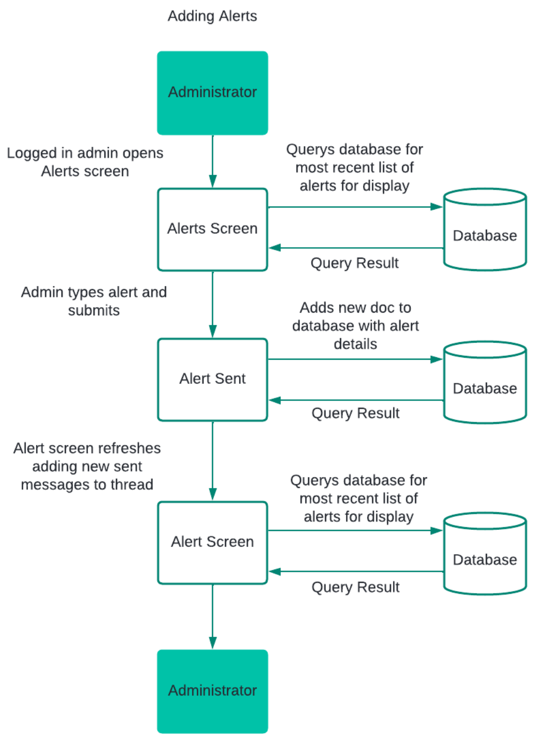
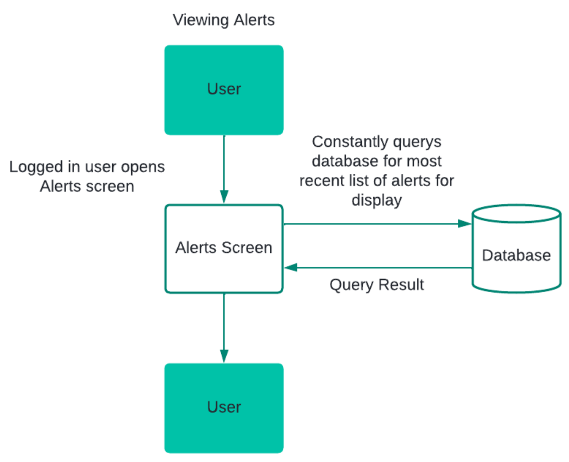
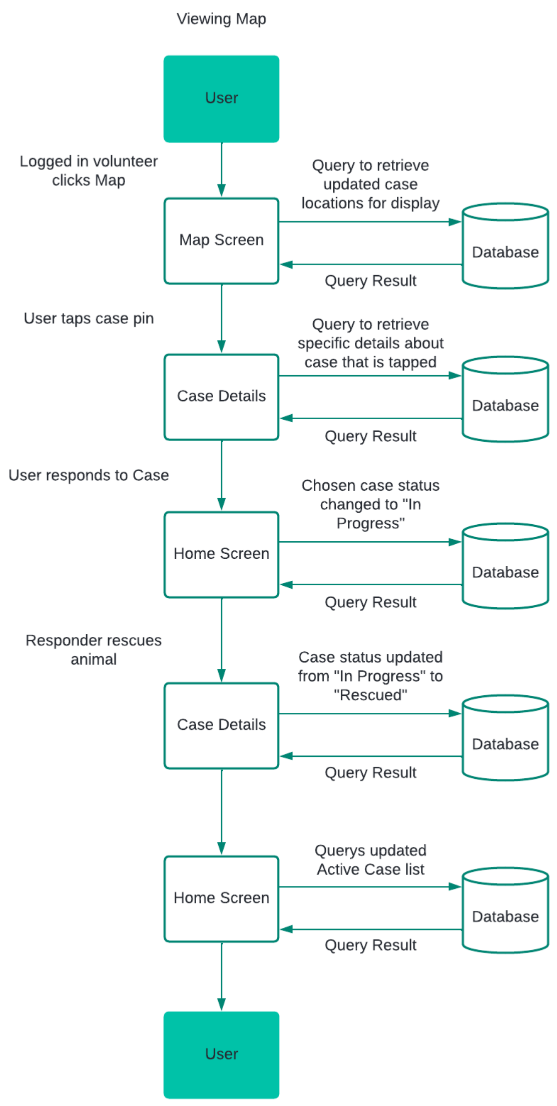

Adding Alerts
Administrator
Logged in admin opens Alerts screen
Alerts Screen
Querys database for most recent list of alerts for display
Database
Query Result
Admin types alert and submits
Alert Sent
Adds new doc to database with alert details
Database
Query Result
Alert screen refreshes adding new sent messages to thread
Alert Screen
Querys database for most recent list of alerts for display
Database
Query Result
Administrator

Figure C.3.6 - Adding Alerts Dataflow



Viewing Alerts
User
Logged in user opens Alerts screen
Alerts Screen
Constantly querys database for most recent list of alerts for display
Database
Query Result
User

Figure C.3.7 - Viewing Alerts Dataflow



Viewing Map
User
Logged in volunteer clicks Map
Map Screen
Query to retrieve updated case locations for display
Database
Query Result
User taps case pin
Case Details
Query to retrieve specific details about case that is tapped
Database
Query Result
User responds to Case
Home Screen
Chosen case status changed to "In Progress"
Database
Query Result
Responder rescues animal
Case Details
Case status updated from "In Progress" to "Rescued"
Database
Query Result
Home Screen
Querys updated Active Case list
Database
Query Result
User

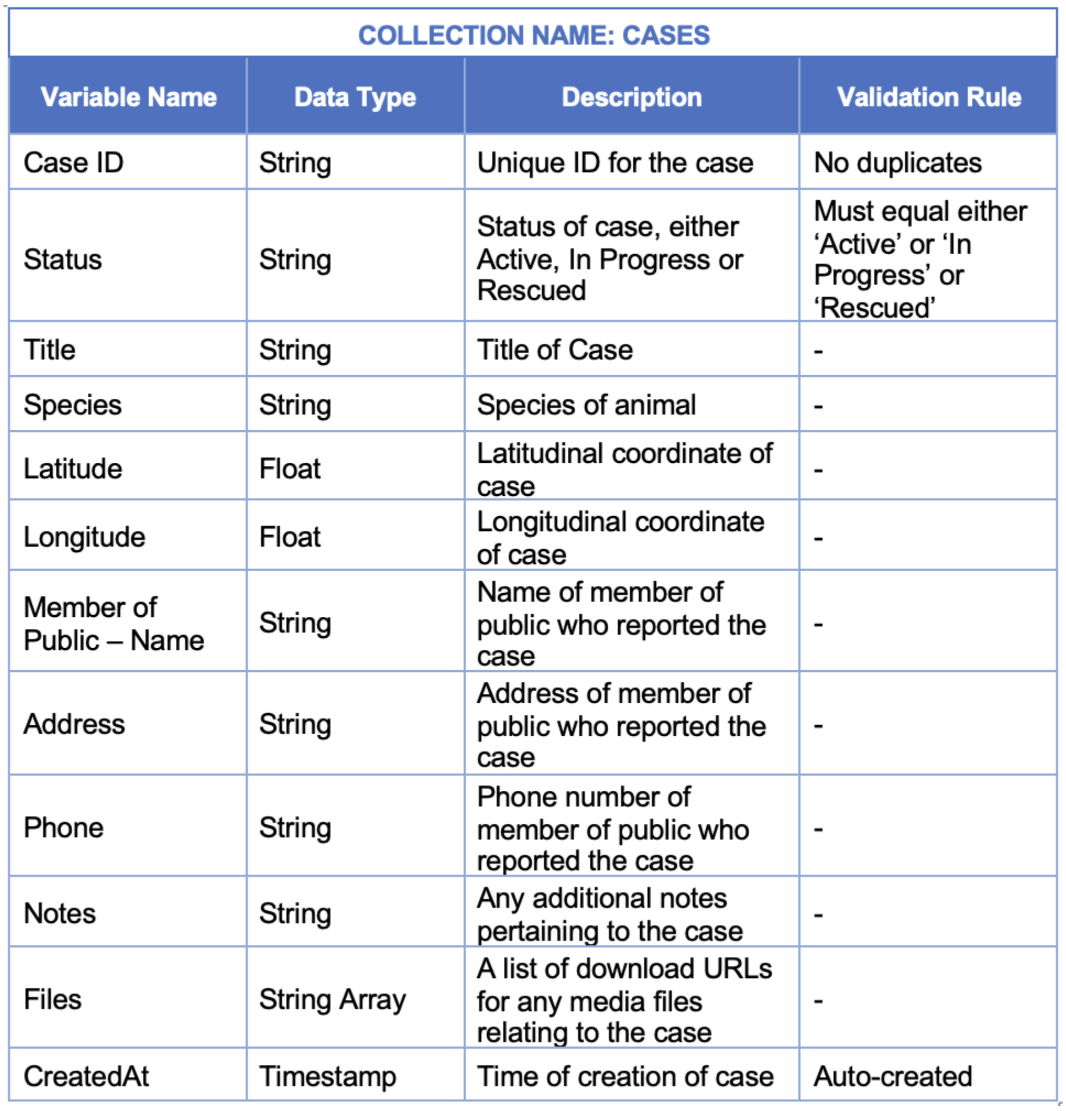
Figure C.3.8 - Map View Dataflow

C.4 - Database Structure

Due to the nature of the application, as well as the benefits of using Firebase as a Database Manager, the developer was afforded the luxury of skipping many of the more complex database design steps which may otherwise have been necessary. Unlike relational databases, Cloud Firestore does not manage tables, but instead uses collections to store data as JSON-like objects. Inside these collections are documents, which can store a wide variety of different attributes and data types. Documents may also store subcollections, which allows the developer to build hierarchical data structures which can scale as the application grows with ease. Firestore also allows for storage of multi-valued items such as arrays and nested objects. This may cause a range of complexities for a more extensive database, however for this system it provides many benefits.

The tables below show the collections stored in the Cloud Firestore database. As described above, due to the fact that this is not a relational database, many complexities and complications are avoided and the database is kept relatively simple. While Firestore does support querying and filtering of data, it does not enforce strict data schemas or relationships between tables, which is a key characteristic of relational databases. These factors provided several benefits for this system.

Table C.4.1 - Cases Collection



COLLECTION NAME: CASES			
Variable Name	**Data Type**	**Description**	**Validation Rule**
Case ID	String	Unique ID for the case	No duplicates
Status	String	Status of case, either Active, In Progress or Rescued	Must equal either 'Active' or 'In Progress' or 'Rescued'
Title	String	Title of Case	-
Species	String	Species of animal	-
Latitude	Float	Latitudinal coordinate of case	-
Longitude	Float	Longitudinal coordinate of case	-
Member of Public – Name	String	Name of member of public who reported the case	-
Address	String	Address of member of public who reported the case	-
Phone	String	Phone number of member of public who reported the case	-
Notes	String	Any additional notes pertaining to the case	-
Files	String Array	A list of download URLs for any media files relating to the case	-
CreatedAt	Timestamp	Time of creation of case	Auto-created

Table C.4.2 - Users Collection

COLLECTION NAME: USERS			
Variable Name	**Data Type**	**Description**	**Validation Rule**
User ID	String	Unique ID for the user	No duplicates
Display Name	String	Display Name of the User	-
Email	String	Email Address of the user	Must be a valid email address
Profile Photo URL	String	A download URL pointing to a file in Firebase Cloud Storage	-
User Type	Float	The type of user, either 'Admin' or 'User' for the regular volunteers	Must equal to either 'Admin' or 'User'

Table C.4.3 - Alerts Collection

COLLECTION NAME: ALERTS			
Variable Name	**Data Type**	**Description**	**Validation Rule**
Alert ID	String	Unique ID for the alert	No duplicates
Text	String	Text written by administrators for the alert	-
CreatedAt	Timestamp	Time of creation of the alert	Auto-created

Table C.4.4 - Messages Collection

COLLECTION NAME: MESSAGES				
Variable Name		**Data Type**	**Description**	**Validation Rule**
Message ID		String	Unique ID for the alert	No duplicates
Text		String	Text of the message written by the user	-
CreatedAt		Timestamp	Time of sending of the message	Auto-created
User	ID	String	ID of User who sent the message	-
	Photo URL	String	Profile Photo URL of user who sent the message	-

C.5 - Activity Flow Diagrams

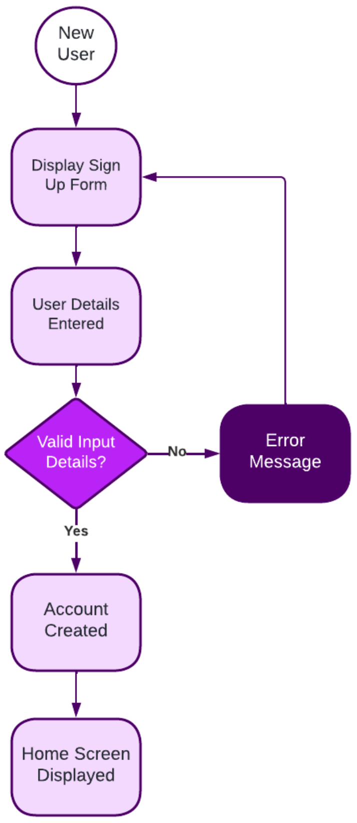


Figure C.5.1 - Sign Up Activity Flow

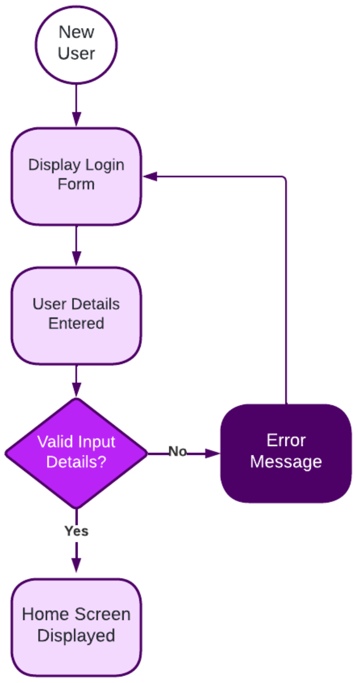


Figure C.5.2 - Login Activity Flow

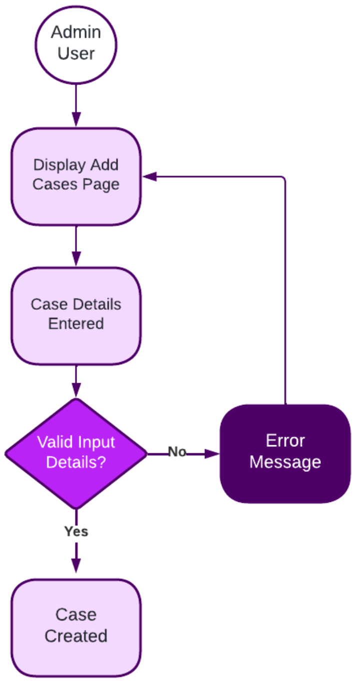


Figure C.5.3 - Add Case Activity Flow

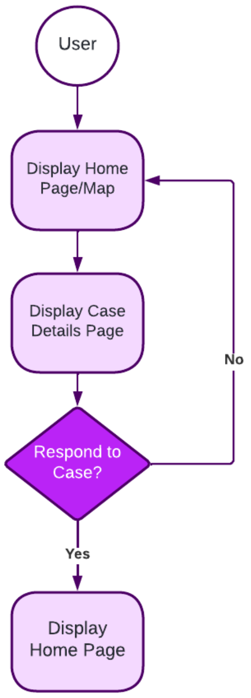
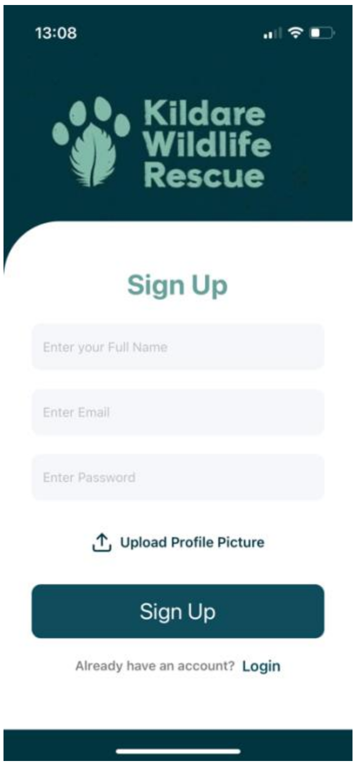


Figure C.5.4 - Respond to Case Activity Flow

APPENDIX D - SYSTEM SCREENS


13:08

Kildare Wildlife Rescue

Sign Up

Enter your Full Name

Enter Email

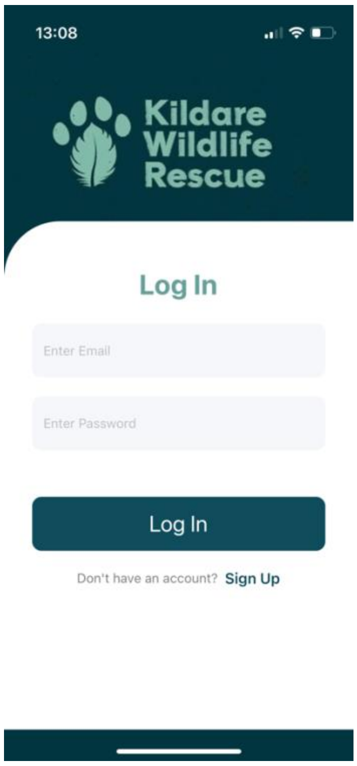
Enter Password

Upload Profile Picture

Sign Up

Already have an account? Login

Figure D.1 – Sign Up Screen


13:08

Kildare Wildlife Rescue

Log In

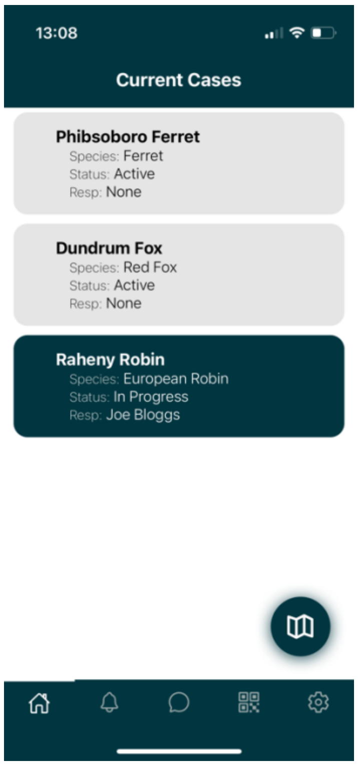
Enter Email

Enter Password

Log In

Don't have an account? Sign Up

Figure D.2 – Login Screen


13:08

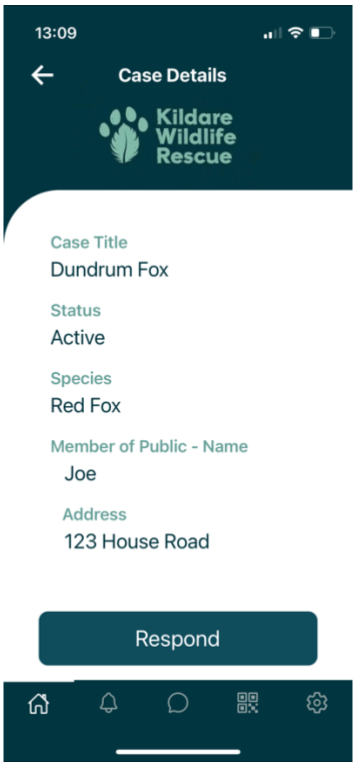
Current Cases

Phibsoboro Ferret
Species: Ferret
Status: Active
Resp: None

Dundrum Fox
Species: Red Fox
Status: Active
Resp: None

Raheny Robin
Species: European Robin
Status: In Progress
Resp: Joe Bloggs

Figure D.3 – Home Screen


13:09

Case Details

Kildare Wildlife Rescue

Case Title
Dundrum Fox

Status
Active

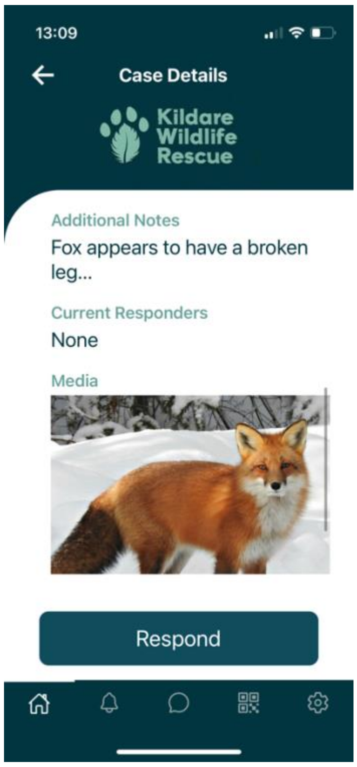
Species
Red Fox

Member of Public - Name
Joe

Address
123 House Road

Respond

Figure D.4 – Case Details


13:09

Case Details

Kildare Wildlife Rescue

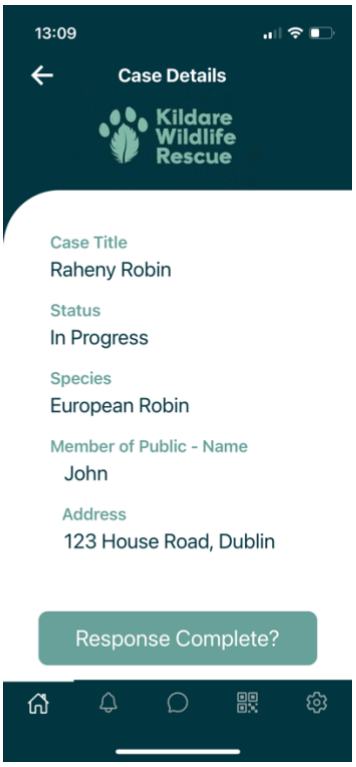
Additional Notes
Fox appears to have a broken leg...

Current Responders
None

Media

Respond

Figure D.5 – Case Details cont'd


13:09

Case Details

Kildare Wildlife Rescue

Case Title
Raheny Robin

Status
In Progress

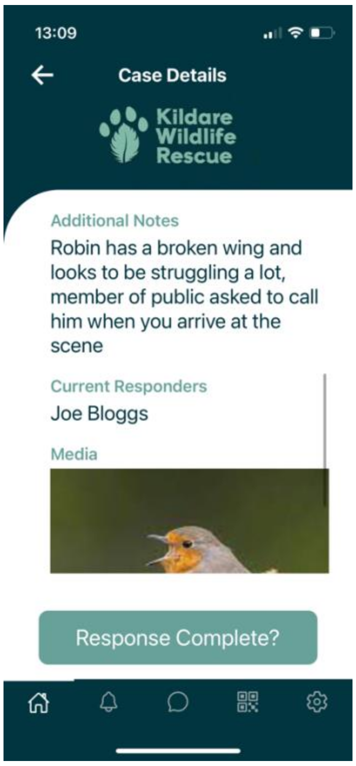
Species
European Robin

Member of Public - Name
John

Address
123 House Road, Dublin

Response Complete?

Figure D.6 – Case Details (Responder)


13:09

Case Details

Kildare Wildlife Rescue

Additional Notes

Robin has a broken wing and looks to be struggling a lot, member of public asked to call him when you arrive at the scene

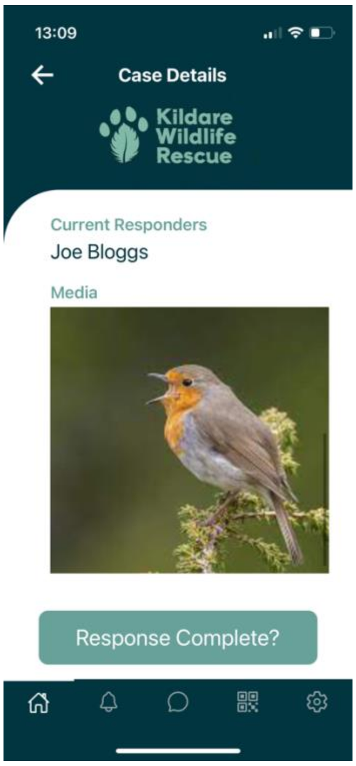
Current Responders

Joe Bloggs

Media

Response Complete?

Figure D.7 – Case Details (Responder) cont'd


13:09

Case Details

Kildare Wildlife Rescue

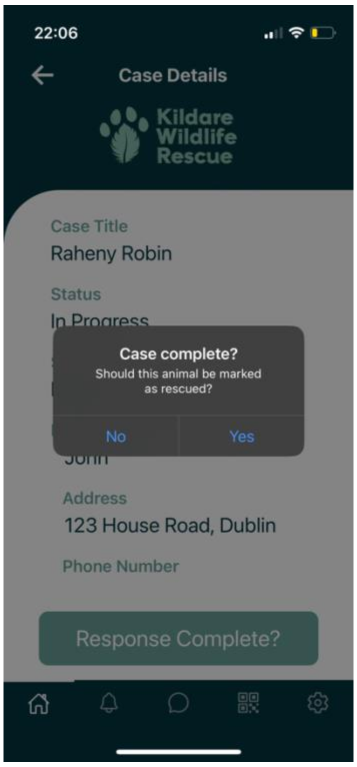
Current Responders

Joe Bloggs

Media

Response Complete?

Figure D.8 – Case Details (Responder) cont'd


22:06

Case Details

Kildare Wildlife Rescue

Case Title

Raheny Robin

Status

In Progress

Case complete?

Should this animal be marked as rescued?

No Yes

John

Address

123 House Road, Dublin

Phone Number

Response Complete?

Figure D.9 – Case Complete Alert

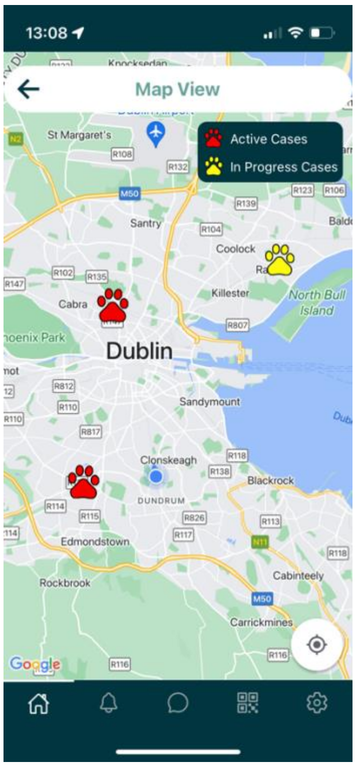
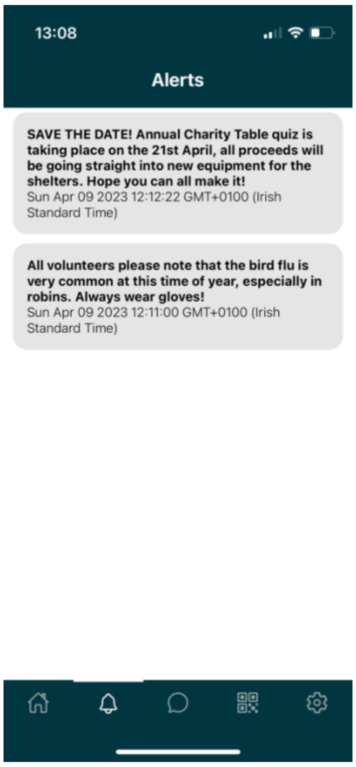


Figure D.10 – Map Screen

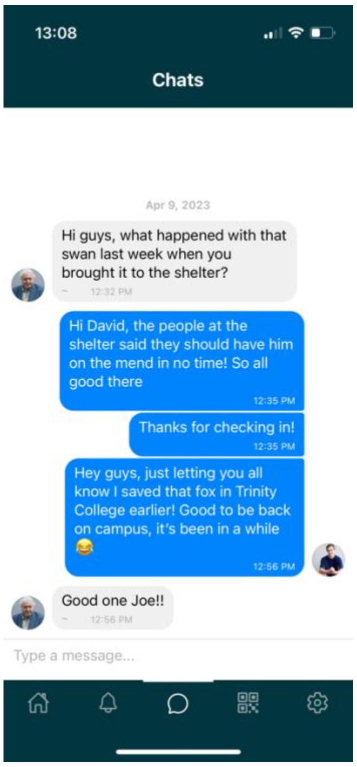

13:08

Alerts

SAVE THE DATE! Annual Charity Table quiz is taking place on the 21st April, all proceeds will be going straight into new equipment for the shelters. Hope you can all make it!
Sun Apr 09 2023 12:12:22 GMT+0100 (Irish Standard Time)

All volunteers please note that the bird flu is very common at this time of year, especially in robins. Always wear gloves!
Sun Apr 09 2023 12:11:00 GMT+0100 (Irish Standard Time)

Figure D.11 – Alerts Screen


13:08

Chats

Apr 9, 2023

Hi guys, what happened with that swan last week when you brought it to the shelter?
12:32 PM

Hi David, the people at the shelter said they should have him on the mend in no time! So all good there
12:35 PM

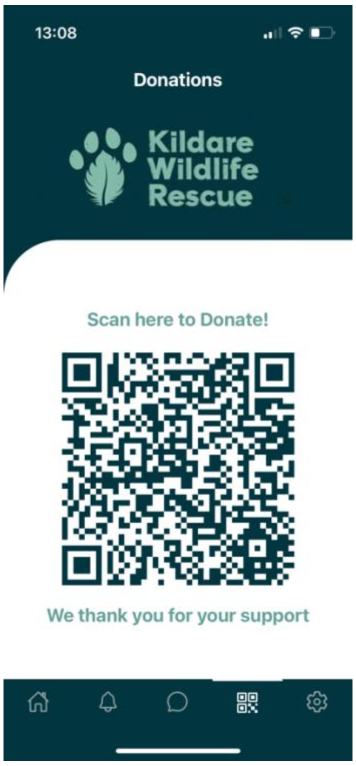
Thanks for checking in!
12:35 PM

Hey guys, just letting you all know I saved that fox in Trinity College earlier! Good to be back on campus, it's been in a while 😂
12:56 PM

Good one Joe!!
12:56 PM

Type a message...

Figure D.12 – Chat Screen


13:08

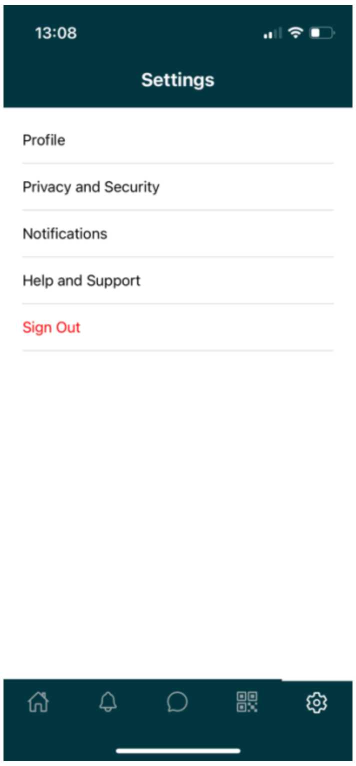
Donations

Kildare Wildlife Rescue

Scan here to Donate!

We thank you for your support

Figure D.13 – Donations Screen


13:08

Settings

Profile

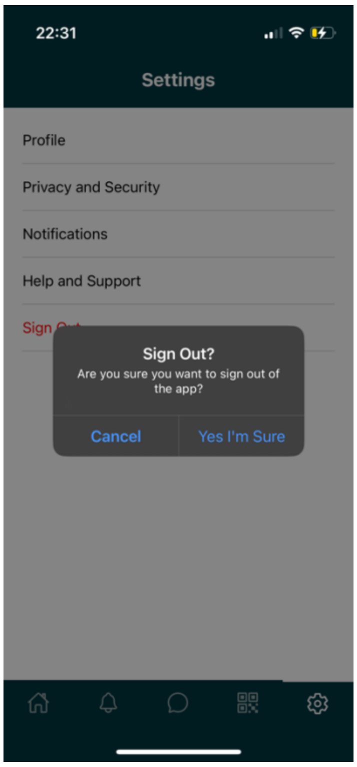
Privacy and Security

Notifications

Help and Support

Sign Out

Figure D.14 – Settings Screen


22:31

Settings

Profile

Privacy and Security

Notifications

Help and Support

Sign Out

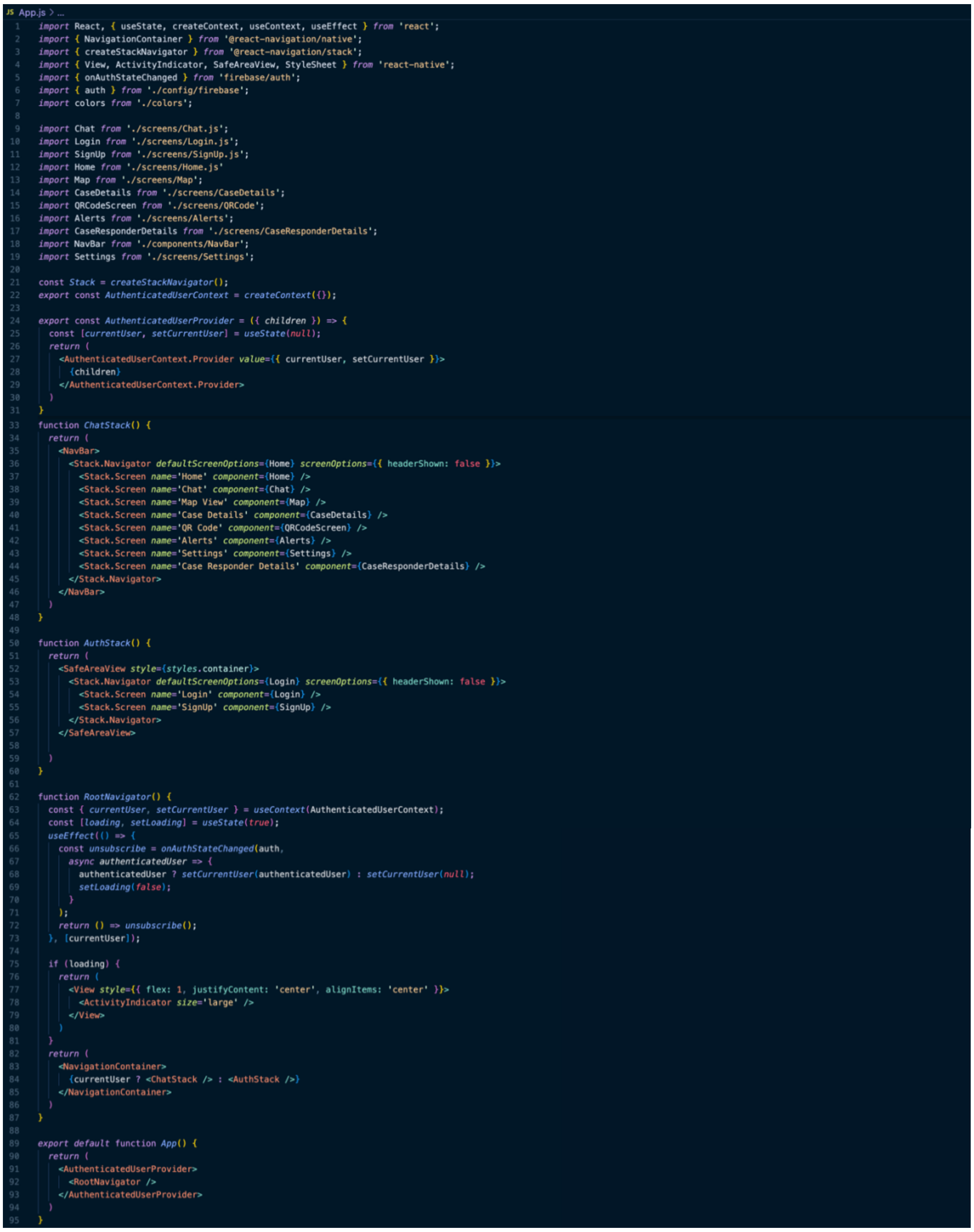
Sign Out?

Are you sure you want to sign out of the app?

Cancel Yes I'm Sure

Figure D.15 –Sign Out Alert

APPENDIX E - SAMPLE SOURCE CODE

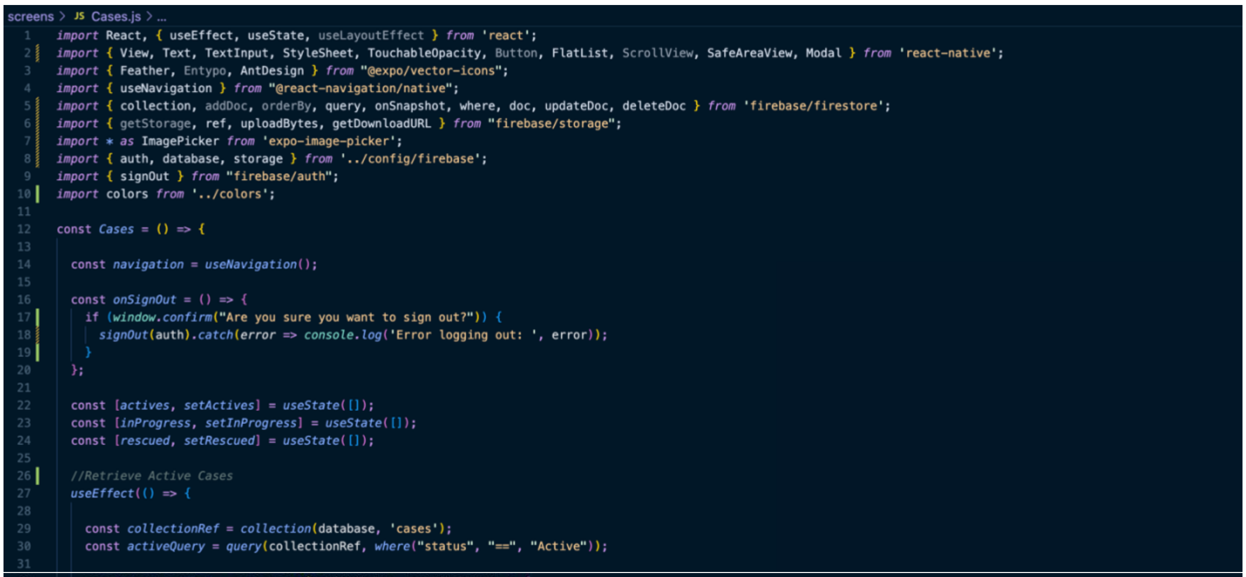


App.js > ...

```
import React, { useState, createContext, useContext, useEffect } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { View, ActivityIndicator, SafeAreaView, StyleSheet } from 'react-native';
import { onAuthStateChanged } from 'firebase/auth';
import { auth } from './config/firebase';
import colors from './colors';

import Chat from './screens/Chat.js';
import Login from './screens/Login.js';
import SignUp from './screens/SignUp.js';
import Home from './screens/Home.js'
import Map from './screens/Map';
import CaseDetails from './screens/CaseDetails';
import QRCodeScreen from './screens/QRCode';
import Alerts from './screens/Alerts';
import CaseResponderDetails from './screens/CaseResponderDetails';
import NavBar from './components/NavBar';
import Settings from './screens/Settings';

const Stack = createStackNavigator();
export const AuthenticatedUserContext = createContext({});

export const AuthenticatedUserProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  return (
    <AuthenticatedUserContext.Provider value={{ currentUser, setCurrentUser }}>
      {children}
    </AuthenticatedUserContext.Provider>
  )
}
function ChatStack() {
  return (
    <NavBar>
      <Stack.Navigator defaultScreenOptions={Home} screenOptions={{ headerShown: false }}>
        <Stack.Screen name='Home' component={Home} />
        <Stack.Screen name='Chat' component={Chat} />
        <Stack.Screen name='Map View' component={Map} />
        <Stack.Screen name='Case Details' component={CaseDetails} />
        <Stack.Screen name='QR Code' component={QRCodeScreen} />
        <Stack.Screen name='Alerts' component={Alerts} />
        <Stack.Screen name='Settings' component={Settings} />
        <Stack.Screen name='Case Responder Details' component={CaseResponderDetails} />
      </Stack.Navigator>
    </NavBar>
  )
}

function AuthStack() {
  return (
    <SafeAreaView style={styles.container}>
      <Stack.Navigator defaultScreenOptions={Login} screenOptions={{ headerShown: false }}>
        <Stack.Screen name='Login' component={Login} />
        <Stack.Screen name='SignUp' component={SignUp} />
      </Stack.Navigator>
    </SafeAreaView>

  )
}

function RootNavigator() {
  const { currentUser, setCurrentUser } = useContext(AuthenticatedUserContext);
  const [loading, setLoading] = useState(true);
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth,
      async authenticatedUser => {
        authenticatedUser ? setCurrentUser(authenticatedUser) : setCurrentUser(null);
        setLoading(false);
      }
    );
    return () => unsubscribe();
  }, [currentUser]);

  if (loading) {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <ActivityIndicator size='large' />
      </View>
    )
  }
  return (
    <NavigationContainer>
      {currentUser ? <ChatStack /> : <AuthStack />}
    </NavigationContainer>
  )
}

export default function App() {
  return (
    <AuthenticatedUserProvider>
      <RootNavigator />
    </AuthenticatedUserProvider>
  )
}
```

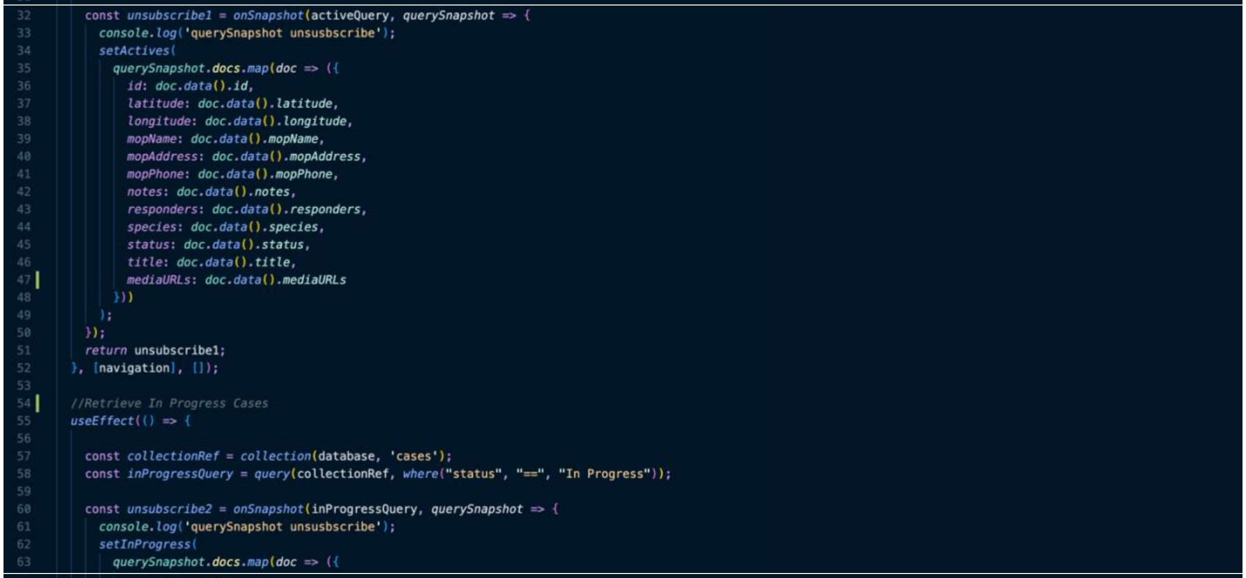
Figure E.1.1 – App.js file from the Mobile Application

The above code is for the App.js file in the mobile application, and shows the structure of the app. Two navigation stacks are created, an Auth Stack and a Chat stack, and depending on whether the user is logged in or not they will be directed to one or the other. Note the Chat Stack is wrapped in a 'NavBar' component, allowing for the Navigation Bar to be visible on every screen in the stack.

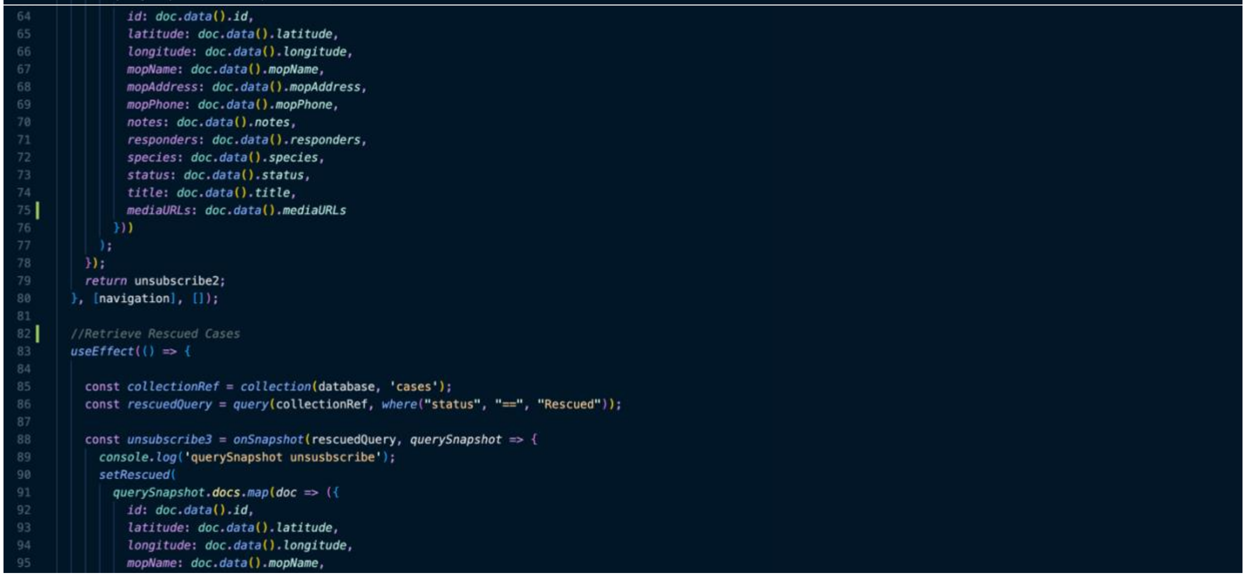


screens > JS Cases.js > ...

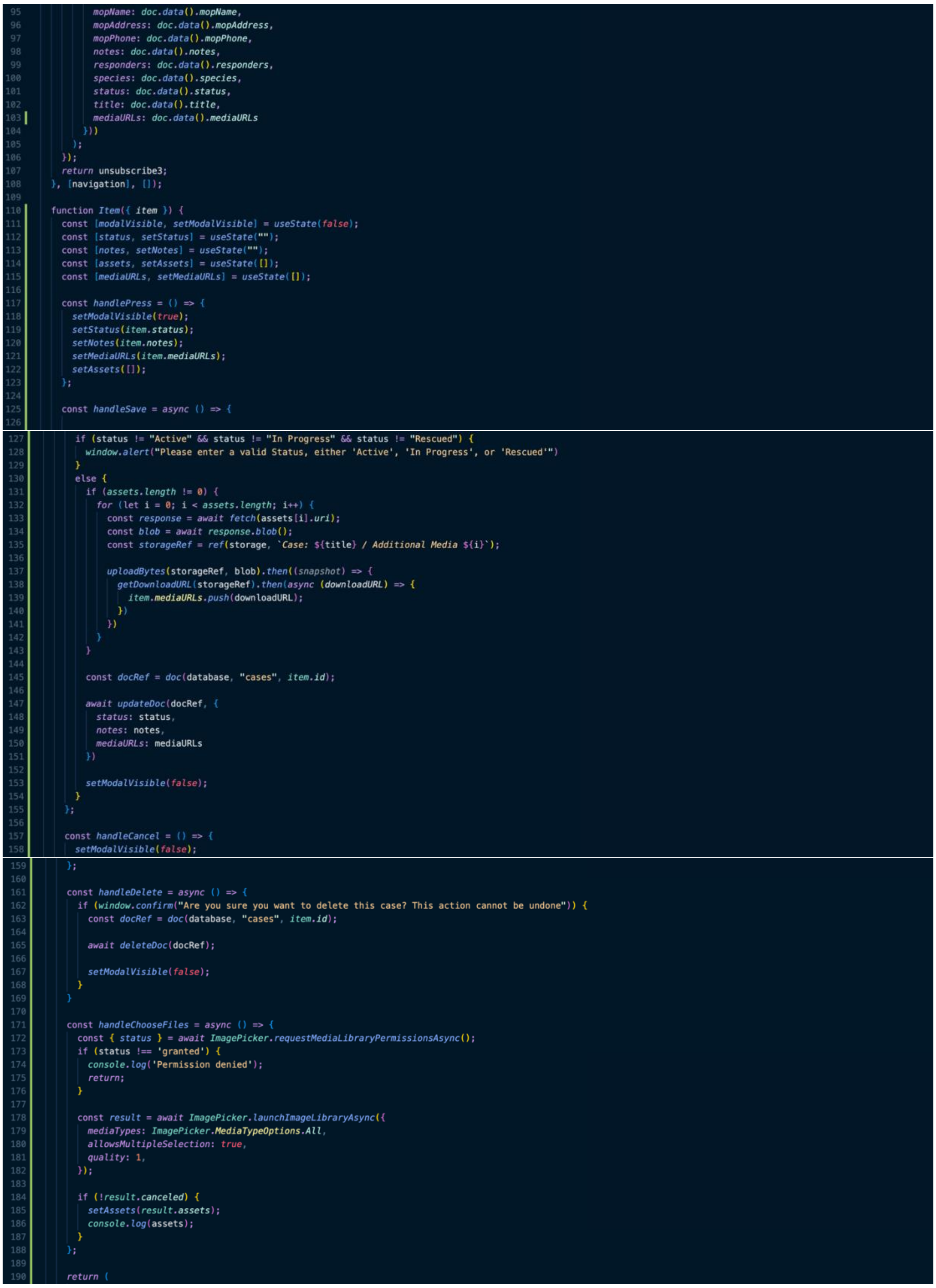
```
import React, { useEffect, useState, useLayoutEffect } from 'react';
import { View, Text, TextInput, StyleSheet, TouchableOpacity, Button, FlatList, ScrollView, SafeAreaView, Modal } from 'react-native';
import { Feather, Entypo, AntDesign } from "@expo/vector-icons";
import { useNavigation } from "@react-navigation/native";
import { collection, addDoc, orderBy, query, onSnapshot, where, doc, updateDoc, deleteDoc } from 'firebase/firestore';
import { getStorage, ref, uploadBytes, getDownloadURL } from "firebase/storage";
import * as ImagePicker from 'expo-image-picker';
import { auth, database, storage } from '../config/firebase';
import { signOut } from "firebase/auth";
import colors from '../colors';

const Cases = () => {

  const navigation = useNavigation();

  const onSignOut = () => {
    if (window.confirm("Are you sure you want to sign out?")) {
      signOut(auth).catch(error => console.log('Error logging out: ', error));
    }
  };

  const [actives, setActives] = useState([]);
  const [inProgress, setInProgress] = useState([]);
  const [rescued, setRescued] = useState([]);

  //Retrieve Active Cases
  useEffect(() => {

    const collectionRef = collection(database, 'cases');
    const activeQuery = query(collectionRef, where("status", "==", "Active"));

```



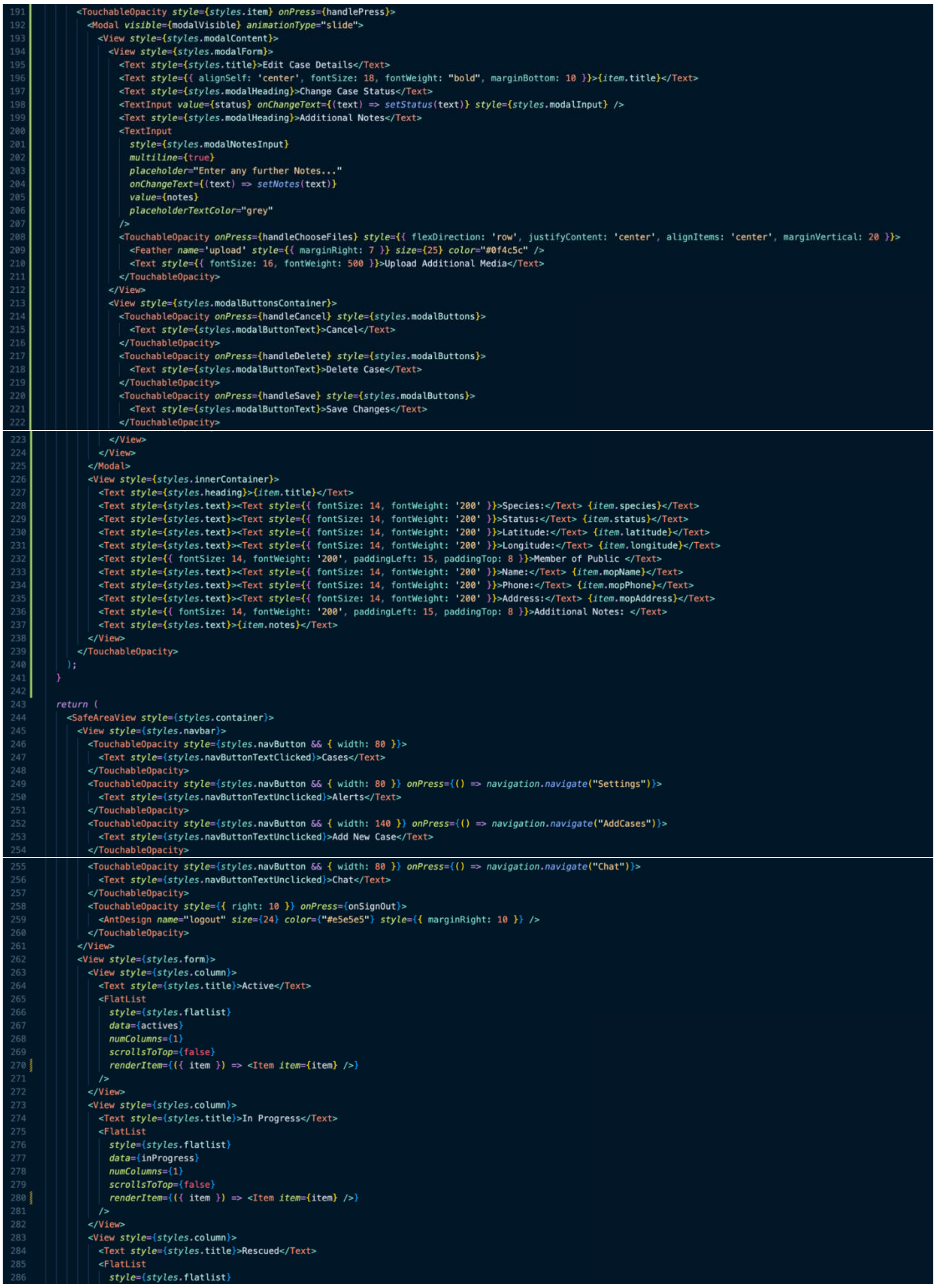
```
    const unsubscribe1 = onSnapshot(activeQuery, querySnapshot => {
      console.log('querySnapshot unsusbscribe');
      setActives(
        querySnapshot.docs.map(doc => ({
          id: doc.data().id,
          latitude: doc.data().latitude,
          longitude: doc.data().longitude,
          mopName: doc.data().mopName,
          mopAddress: doc.data().mopAddress,
          mopPhone: doc.data().mopPhone,
          notes: doc.data().notes,
          responders: doc.data().responders,
          species: doc.data().species,
          status: doc.data().status,
          title: doc.data().title,
          mediaURLs: doc.data().mediaURLs
        }))
      );
    });
    return unsubscribe1;
  }, [navigation], []);

  //Retrieve In Progress Cases
  useEffect(() => {

    const collectionRef = collection(database, 'cases');
    const inProgressQuery = query(collectionRef, where("status", "==", "In Progress"));

    const unsubscribe2 = onSnapshot(inProgressQuery, querySnapshot => {
      console.log('querySnapshot unsusbscribe');
      setInProgress(
        querySnapshot.docs.map(doc => ({
```



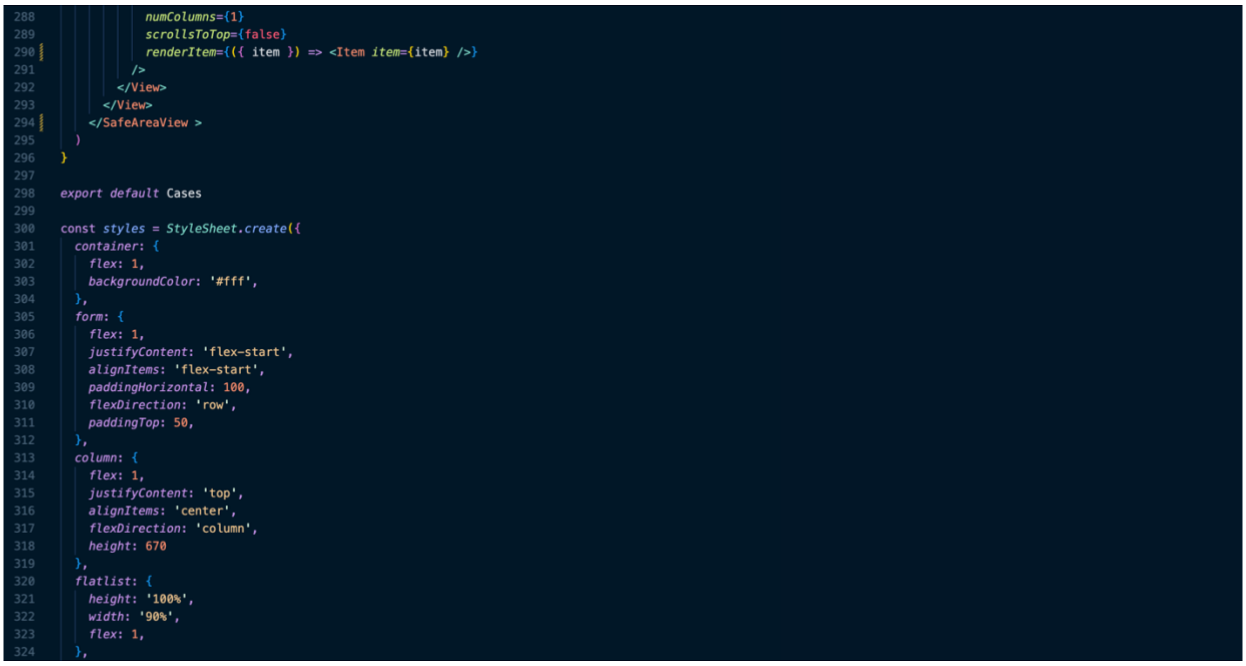
```
          id: doc.data().id,
          latitude: doc.data().latitude,
          longitude: doc.data().longitude,
          mopName: doc.data().mopName,
          mopAddress: doc.data().mopAddress,
          mopPhone: doc.data().mopPhone,
          notes: doc.data().notes,
          responders: doc.data().responders,
          species: doc.data().species,
          status: doc.data().status,
          title: doc.data().title,
          mediaURLs: doc.data().mediaURLs
        }))
      );
    });
    return unsubscribe2;
  }, [navigation], []);

  //Retrieve Rescued Cases
  useEffect(() => {

    const collectionRef = collection(database, 'cases');
    const rescuedQuery = query(collectionRef, where("status", "==", "Rescued"));

    const unsubscribe3 = onSnapshot(rescuedQuery, querySnapshot => {
      console.log('querySnapshot unsusbscribe');
      setRescued(
        querySnapshot.docs.map(doc => ({
          id: doc.data().id,
          latitude: doc.data().latitude,
          longitude: doc.data().longitude,
          mopName: doc.data().mopName,
```



```javascript
        mopName: doc.data().mopName,
        mopAddress: doc.data().mopAddress,
        mopPhone: doc.data().mopPhone,
        notes: doc.data().notes,
        responders: doc.data().responders,
        species: doc.data().species,
        status: doc.data().status,
        title: doc.data().title,
        mediaURLs: doc.data().mediaURLs
      }))
    );
  });
  return unsubscribe3;
}, [navigation], []);

function Item({ item }) {
  const [modalVisible, setModalVisible] = useState(false);
  const [status, setStatus] = useState("");
  const [notes, setNotes] = useState("");
  const [assets, setAssets] = useState([]);
  const [mediaURLs, setMediaURLs] = useState([]);

  const handlePress = () => {
    setModalVisible(true);
    setStatus(item.status);
    setNotes(item.notes);
    setMediaURLs(item.mediaURLs);
    setAssets([]);
  };

  const handleSave = async () => {

    if (status != "Active" && status != "In Progress" && status != "Rescued") {
      window.alert("Please enter a valid Status, either 'Active', 'In Progress', or 'Rescued'")
    }
    else {
      if (assets.length != 0) {
        for (let i = 0; i < assets.length; i++) {
          const response = await fetch(assets[i].uri);
          const blob = await response.blob();
          const storageRef = ref(storage, `Case: ${title} / Additional Media ${i}`);

          uploadBytes(storageRef, blob).then((snapshot) => {
            getDownloadURL(storageRef).then(async (downloadURL) => {
              item.mediaURLs.push(downloadURL);
            })
          })
        }
      }

      const docRef = doc(database, "cases", item.id);

      await updateDoc(docRef, {
        status: status,
        notes: notes,
        mediaURLs: mediaURLs
      })

      setModalVisible(false);
    }
  };

  const handleCancel = () => {
    setModalVisible(false);
  };

  const handleDelete = async () => {
    if (window.confirm("Are you sure you want to delete this case? This action cannot be undone")) {
      const docRef = doc(database, "cases", item.id);

      await deleteDoc(docRef);

      setModalVisible(false);
    }
  }

  const handleChooseFiles = async () => {
    const { status } = await ImagePicker.requestMediaLibraryPermissionsAsync();
    if (status !== 'granted') {
      console.log('Permission denied');
      return;
    }

    const result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.All,
      allowsMultipleSelection: true,
      quality: 1,
    });

    if (!result.canceled) {
      setAssets(result.assets);
      console.log(assets);
    }
  };

  return (
```



```jsx
      <TouchableOpacity style={styles.item} onPress={handlePress}>
        <Modal visible={modalVisible} animationType="slide">
          <View style={styles.modalContent}>
            <View style={styles.modalForm}>
              <Text style={styles.title}>Edit Case Details</Text>
              <Text style={{ alignSelf: 'center', fontSize: 18, fontWeight: "bold", marginBottom: 10 }}>{item.title}</Text>
              <Text style={styles.modalHeading}>Change Case Status</Text>
              <TextInput value={status} onChangeText={(text) => setStatus(text)} style={styles.modalInput} />
              <Text style={styles.modalHeading}>Additional Notes</Text>
              <TextInput
                style={styles.modalNotesInput}
                multiline={true}
                placeholder="Enter any further Notes..."
                onChangeText={(text) => setNotes(text)}
                value={notes}
                placeholderTextColor="grey"
              />
              <TouchableOpacity onPress={handleChooseFiles} style={{ flexDirection: 'row', justifyContent: 'center', alignItems: 'center', marginVertical: 20 }}>
                <Feather name='upload' style={{ marginRight: 7 }} size={25} color="#0f4c5c" />
                <Text style={{ fontSize: 16, fontWeight: 500 }}>Upload Additional Media</Text>
              </TouchableOpacity>
            </View>
            <View style={styles.modalButtonsContainer}>
              <TouchableOpacity onPress={handleCancel} style={styles.modalButtons}>
                <Text style={styles.modalButtonText}>Cancel</Text>
              </TouchableOpacity>
              <TouchableOpacity onPress={handleDelete} style={styles.modalButtons}>
                <Text style={styles.modalButtonText}>Delete Case</Text>
              </TouchableOpacity>
              <TouchableOpacity onPress={handleSave} style={styles.modalButtons}>
                <Text style={styles.modalButtonText}>Save Changes</Text>
              </TouchableOpacity>
            </View>
          </View>
        </Modal>
        <View style={styles.innerContainer}>
          <Text style={styles.heading}>{item.title}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Species:</Text> {item.species}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Status:</Text> {item.status}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Latitude:</Text> {item.latitude}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Longitude:</Text> {item.longitude}</Text>
          <Text style={{ fontSize: 14, fontWeight: '200', paddingLeft: 15, paddingTop: 8 }}>Member of Public </Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Name:</Text> {item.mopName}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Phone:</Text> {item.mopPhone}</Text>
          <Text style={styles.text}><Text style={{ fontSize: 14, fontWeight: '200' }}>Address:</Text> {item.mopAddress}</Text>
          <Text style={{ fontSize: 14, fontWeight: '200', paddingLeft: 15, paddingTop: 8 }}>Additional Notes: </Text>
          <Text style={styles.text}>{item.notes}</Text>
        </View>
      </TouchableOpacity>
    );
  }

  return (
    <SafeAreaView style={styles.container}>
      <View style={styles.navbar}>
        <TouchableOpacity style={styles.navButton && { width: 80 }}>
          <Text style={styles.navButtonTextClicked}>Cases</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.navButton && { width: 80 }} onPress={() => navigation.navigate("Settings")}>
          <Text style={styles.navButtonTextUnclicked}>Alerts</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.navButton && { width: 140 }} onPress={() => navigation.navigate("AddCases")}>
          <Text style={styles.navButtonTextUnclicked}>Add New Case</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.navButton && { width: 80 }} onPress={() => navigation.navigate("Chat")}>
          <Text style={styles.navButtonTextUnclicked}>Chat</Text>
        </TouchableOpacity>
        <TouchableOpacity style={{ right: 10 }} onPress={onSignOut}>
          <AntDesign name="logout" size={24} color={"#e5e5e5"} style={{ marginRight: 10 }} />
        </TouchableOpacity>
      </View>
      <View style={styles.form}>
        <View style={styles.column}>
          <Text style={styles.title}>Active</Text>
          <FlatList
            style={styles.flatlist}
            data={actives}
            numColumns={1}
            scrollsToTop={false}
            renderItem={({ item }) => <Item item={item} />}
          />
        </View>
        <View style={styles.column}>
          <Text style={styles.title}>In Progress</Text>
          <FlatList
            style={styles.flatlist}
            data={inProgress}
            numColumns={1}
            scrollsToTop={false}
            renderItem={({ item }) => <Item item={item} />}
          />
        </View>
        <View style={styles.column}>
          <Text style={styles.title}>Rescued</Text>
          <FlatList
            style={styles.flatlist}
```



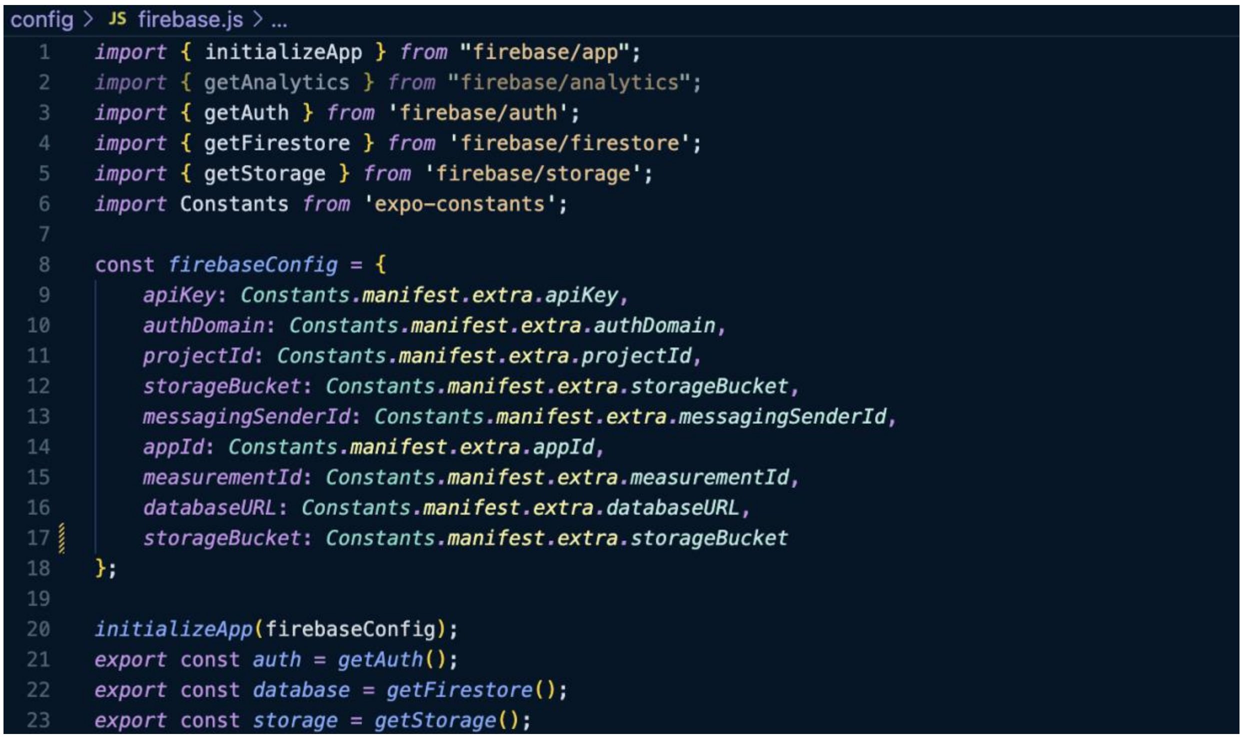
```
            numColumns={1}
            scrollsToTop={false}
            renderItem={({ item }) => <Item item={item} />}
          />
        </View>
      </View>
    </SafeAreaView >
  )
}

export default Cases

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
  },
  form: {
    flex: 1,
    justifyContent: 'flex-start',
    alignItems: 'flex-start',
    paddingHorizontal: 100,
    flexDirection: 'row',
    paddingTop: 50,
  },
  column: {
    flex: 1,
    justifyContent: 'top',
    alignItems: 'center',
    flexDirection: 'column',
    height: 670
  },
  flatlist: {
    height: '100%',
    width: '90%',
    flex: 1,
  },
```

Figure E.1.2 – Cases.js file, Cases page from the Web Application excluding some styling

This rather lengthy code sample is for the landing page, or Cases page, of the Web Application. It employs the use of React's 'useEffect' Hook in order to keep the case data displayed here up to date. This hook is called every time the page renders, or is refreshed, and will query the database to retrieve the most up to date content.

A 'Modal' component is also used to create the pop-up which allows the administrator to Edit the case details. This modal is controlled using a 'setModalVisible' boolean which is initialised to false and set to true when a case is clicked on.

The StyleSheet component at the very bottom of the code sample allows for other components on the page to be styled with ease. 21 other style types were created and used on this page, however these were cropped out of this sample for length reasons.



```
config > JS firebase.js > ...
 1  import { initializeApp } from "firebase/app";
 2  import { getAnalytics } from "firebase/analytics";
 3  import { getAuth } from 'firebase/auth';
 4  import { getFirestore } from 'firebase/firestore';
 5  import { getStorage } from 'firebase/storage';
 6  import Constants from 'expo-constants';
 7
 8  const firebaseConfig = {
 9      apiKey: Constants.manifest.extra.apiKey,
10      authDomain: Constants.manifest.extra.authDomain,
11      projectId: Constants.manifest.extra.projectId,
12      storageBucket: Constants.manifest.extra.storageBucket,
13      messagingSenderId: Constants.manifest.extra.messagingSenderId,
14      appId: Constants.manifest.extra.appId,
15      measurementId: Constants.manifest.extra.measurementId,
16      databaseURL: Constants.manifest.extra.databaseURL,
17      storageBucket: Constants.manifest.extra.storageBucket
18  };
19
20  initializeApp(firebaseConfig);
21  export const auth = getAuth();
22  export const database = getFirestore();
23  export const storage = getStorage();
```

E.1.3 – firebase.js file, used to connect the Front-End to the Firebase Back-End

The code above shows the Firebase configuration, which allows for the application to communicate with and use the various tools that Firebase has to offer. Note that the various values inside the firebaseConfig constant are hidden as these provide access to the database and so should not be made public. Instead they are declared as extra constants in the app's manifest and imported in line 6. This is discussed further in Section 4.4.6.

APPENDIX F - TEST DOCUMENTATION

This appendix provides an overview of the testing that was carried out on the system during development. Three types of tests were carried out on the system as outlined in *Section 4.5*. Initially Unit tests were carried out to assess individual elements or units of the code to ensure that they are functioning as expected. Examples of the unit tests carried out are listed below.

Table F.1 – Unit Testing Samples

Test	Test Description	Test Results
User Sign Up	Test to establish if account is created and user data is entered into the database correctly when the user Signs Up	Account was created successfully in Firebase Auth and data was added to database in the 'users' collection
User Login (Invalid credentials)	Invalid login details were entered into the login screen	Error message appeared to inform the user that the login details provided were incorrect
User Login (Valid credentials)	Valid login details were entered into the login screen	Login details were accepted and user was logged in successfully
Add Case (Invalid Status)	Added cases with an invalid status, i.e. not "Active", "In Progress" nor "Rescued"	Error message appeared to inform user that an invalid status had been entered
Add Case (Valid Status)	Added cases with a valid status i.e. "Active", "In Progress" and "Rescued"	Case was successfully created and added to the 'cases collection in the database
Navigation Buttons	The navigation buttons in both apps were tested thoroughly to ensure that all navigation routes were working correctly	Each button navigated the user to the correct page
Messages	Messages were sent from devices to ensure that they were adding to the database correctly and also being received by other devices	Messages were sent and received, and added properly to the database
Alerts	Alerts were posted by admin accounts to ensure that they were added to the database properly	Alerts were posted correctly and added to the database
QR Code	The QR Code page of the mobile app was tested to ensure that scanning the code linked to the correct web page	The QR Code linked the charity's Go Fund Me page as expected
Case Response	The "Respond" button was pressed on the Case Details page to check that this was dealt with correctly	The user's name was added to the list of responders and the case status changed to "In Progress"
Logout	The Sign Out button was pressed on both applications to ensure that this was handled correctly	The user was logged out of the application successfully and the unauthenticated user route was returned

Once the individual unit testing had been completed and each component and unit of code was seen to be functioning appropriately, System testing was carried out to ensure that each of these components operated correctly when integrated into the system. Samples of these test can be seen in the below table.

Table F.2 - System Testing Samples

Test	Test Description	Test Results
Protected Route Testing	The Sign Up and Login pages were tested thoroughly to ensure that without valid credentials a user could not gain access to the Protected Route	Only when valid login or sign up details were inputted was the Homepage returned
Case Dataflow	Case dataflows were tested thoroughly to ensure that they were added correctly to the database, updated correctly if edited or responded to and always displayed properly in each application	Cases were added successfully to the database on creation, updated correctly when edited or responded to and always appeared with up to date details in both applications
Alerts Dataflow	Alerts were posted from the web application to ensure they were posting correctly and visible from both applications	Alerts posted as expected and were visible almost instantaneously from the 'Alerts' page of the mobile app
Navigation Bar	The navigation bars were added to each screen of both applications to ensure that the user could navigate fluidly between all screens	The navigation bars worked as expected and the user could navigate between screens with ease

A variety of other tests, both unit and system, were carried out in addition to the examples provided in this section, to ensure that the system was functioning as it was initially intended. Informal user testing was also completed as described in *Section 4.5*, in order to observe and learn from user interaction with the system. All of these tests allowed the developer to make any necessary changes to the system during the development phase and resulted in a fully functioning and fit-for-purpose solution.

APPENDIX G – GLOSSARY

Android: Google's mobile operating system for smartphones, tablets, and other mobile devices, known for its open-source platform and customizable user interface.

API: This stands for Application Programme Interface, a set of rules and protocols that allow different software applications to communicate with each other.

CLI: Stands for Command Line Interface, a way of interacting with a computer program through text commands entered into a terminal, rather than a graphical user interface.

Client-Side: The part of an application or system that runs on the user's device (e.g., web browser), including the user interface and scripts that perform tasks such as form validation and dynamic page updates.

Firebase: A Google-owned mobile and web application development platform that offers backend services such as real-time database, authentication, hosting, and cloud storage, enabling developers to create scalable applications with simple APIs and SDKs.

iOS: Apple's mobile operating system for iPhone, iPad, and iPod Touch, known for its intuitive user interface, security features, and seamless integration with Apple devices and services.

JavaScript: An object-oriented programming language for creating interactive web pages and applications, used on both client-side and server-side.

Mobile Application: A software application designed to run on and accessed through mobile devices. They can be native, built for a specific operating system, or cross-platform, and provide various functionality from games to enterprise applications.

React: A free and open-source JavaScript front-end library for building user interfaces using UI components.

React Native: A free and open-source JavaScript front-end library that allows developers to use the React library to build native mobile apps for iOS, Android, and the web, using a single codebase.

SDK: This stands for Software Development Kit, a set of software tools and resources provided by a software vendor or platform to help developers create applications for a specific platform or operating system.

Server: A computer or computer program which manages access to a centralised resource in a network

Server-Side: The part of an application or website that runs on the server, managing data, processing requests, and generating dynamic content using backend code, database, and APIs.

URL: Stands for Uniform Resource Locator, specifies the unique location a webpage can be found on a server.

Web Application: A software application that (Ullman, 2012)runs on the web, accessed through a web browser. These can be accessed from any device with a web browser and internet connection, and can range from simple static pages to complex, dynamic applications.

REFERENCES

AtomLab, 2022. *Atom Lab.* [Online]
Available at: https://www.atomlab.dev/tutorials/integrating-firebase-react-native
[Accessed 2023].

ExpoDev, 2023. *Expo Dev.* [Online]
Available at: https://expo.dev/
[Accessed 2023].

Google, 2023. *Firebase.* [Online]
Available at: https://firebase.google.com/
[Accessed 2023].

Huls, M., 2021 . *Keep your code secure by using environment variables and env files.* [Online]
Available at: https://towardsdatascience.com/keep-your-code-secure-by-using-environment-variables-and-env-files-4688a70ea286
[Accessed 12 March 2023].

KeeperSecurity, 2022. *Keeper Security.* [Online]
Available at: https://www.keepersecurity.com/blog/2022/09/14/why-is-password-security-important/#:~:text=Importance%20of%20a%20Strong%20Password,for%20both%20businesses%20and%20consumers.
[Accessed 2023].

Manar H. Alalfi, J. R. C. a. T. R. D., 2012. *Recovering Role-Based Access Control Security Models from Dynamic Web Applications.* School of Computing, Queens University, Kingston, Ontario, Canada: s.n.

Medium, 2021. *GitConnected.* [Online]
Available at: https://levelup.gitconnected.com/react-native-cli-vs-expo-cli-which-one-do-i-choose-bdf02ea457bf
[Accessed 2023].

Microsoft, 2023. *Microsoft Security.* [Online]
Available at: https://www.microsoft.com/en-us/security/business/security-101/what-is-password-protection
[Accessed 2023].

Omisola, I., 2021. *Make Use Of.* [Online]
Available at: https://www.makeuseof.com/what-is-google-firebase-why-use-it/
[Accessed 2023].

Powell-Morse, A., 2016. *Iterative Model: What Is It And When Should You Use It?.* [Online]
Available at: https://blog.airbrake.io/blog/sdlc/iterative-model

Ramadan Abdunabi, I. R. R. F., 2013. *Specification and Analysis of Access Control Policies for Mobile Applications.* Colorado State University: s.n.

ReactNative, 2023. *React Native Dev.* [Online]
Available at: https://reactnative.dev/
[Accessed 2023].

Ullman, L., 2012. *Modern JavaScript: Develop and Design.* [Online]
Available at:
https://books.google.ie/books?hl=en&lr=&id=ExJ9_sor87QC&oi=fnd&pg=PR5&dq=book+to+#v=onepage&q=book%20to&f=false
[Accessed 2022].

W3Schools, n.d. *Javascript Conventions.* [Online]
Available at: https://www.w3schools.com/js/js_conventions.asp